CDC 00594

The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! March 1987: Volume 7, Issue 05. $3.50

More Languages

- Programming on the Amiga
- Language Speed Comparisons
- Comparison of C64 'C' Compilers
- CP/M Block Allocation Calculator
- C64 Structured Programming
- Evaluating C-128 CP/M
- Assembling Assemblers
- C-64 23K RamDisk
- Blazin' Forth
- C128 Programmer's Aid

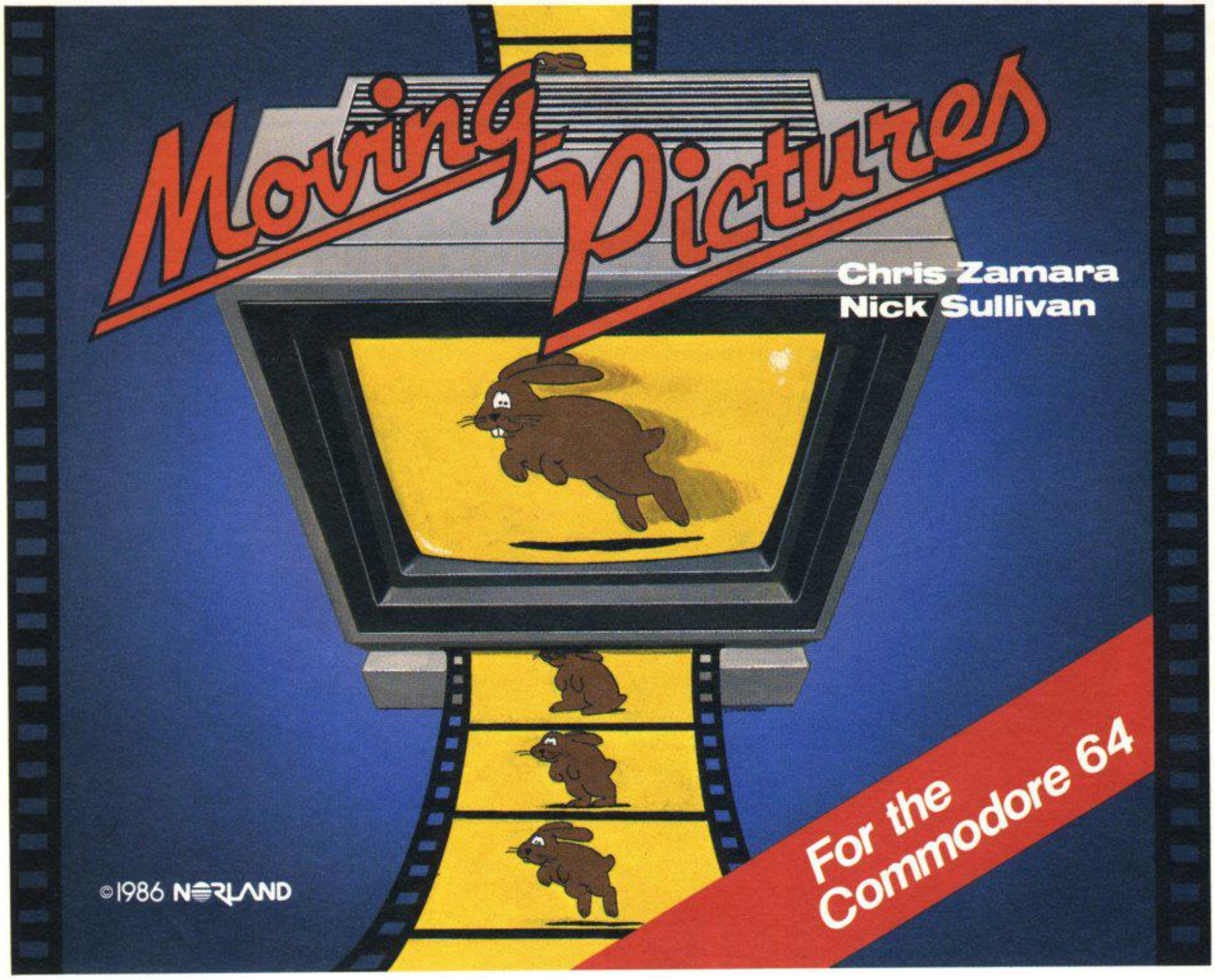
FREE T-Shirt Offer!

see page 77

0 72246 00594 03

J.MOSTACCI

Life Doesn't Stand Still. . .
Why Should Your Pictures?



Full Screen Animation On Your Commodore 64!

It used to be even the experts couldn't do it.
Now, anyone can.
Moving Pictures by AHA! is more than just another animation package. It's a whole new breakthrough in software technology.
Moving Pictures is fast, smooth, full-screen animation that is totally under your control.
You use your favourite graphics tool * to draw the frames of your movie, then show it at full animation speed with a single command!
Write movie "scripts" in BASIC, using the powerful Moving Pictures command set for complete control of your creations!
Whether you're a programmer or a novice, you'll be able to put together and display intricate scenes of your own invention. You can even edit your scripts or execute a BASIC program while a movie is being displayed - Moving Pictures is a **multitasking system!**

Besides being fun in itself, Moving Pictures lets you easily add animation sequences to your own BASIC programs.

Just a few of the many Moving Pictures features:

- allows split screen operation - part graphics, part text - even while a movie is running
- repeat, stop at any frame, change position, and colours, vary display speed and more
- hold several movies in memory and switch instantly from one movie to the other
- instant, on-line help available at the touch of a key
- no copy protection used on the disk
- and here's the best part: the price is just **$29.95**!

* Graphics program not included. Moving Pictures uses a standard hi-res bitmap, so many graphics programs are fully compatible, including: Flexidraw™, Doodle™, Gold Disk Art Package™, Print Shop Screen Magic™, Perspectives™.

Mail Orders: Transactor Publishing Inc., 500 Steeles Avenue, Milton, ON, Canada, L9T 3P7 (416) 878-8438 (or use order card at center).

Canadian and International Dealer Inquiries To:
Norland Software Products, 251 Nipissing Road, Unit 3, Milton, ON, Canada, L9T 4Z5. (416) 876-4774.

USA Dealer Inquiries To:
American Software Distributors Inc., Box 290, Urbana IL, USA 61801 1-800-225-7941.

Volume 7
Issue 05
Circulation at Large
72,000

More Languages

Start Address Editorial 3

Bits and Pieces 6
List 'n' Save
Saving C-128 Function Keys
Dvorak Layout
Finding Relative File Record Lengths
VIC Real-Time Clock Fix
C-128 Program Merge
One-Line Direct-Mode
File Printer
The 1541 Interleave Factor
Fixes For The Compressor
C-64 Time of Day Clocks
Screensave for the C64

Letters 11
Copy-All 64 and relative files:
Adapting Search and Print
Converting to Merlin
'With permission from the T.'
Plus/4's in our future?
Plus/4 anguish, addressed to Jim Butterfield
ML column bandwagon grows
In search of PET classics
More GAMES feedback
Unassembler and Symass fixes
Symass POKE discrepancy
1541 upgrade ROMs
Super Kit - the dark side
Squashing C-64 RS232 bugs

News BRK 77
Submitting NEWS BRK Press Releases
Subscription Intersection Set
No More GLINKS
Schedule IRQ
Late Note
Superpaks from Digital Solutions
Free Transactor T's with Mag+Disk Subscription
Transactor Disk Price Increase
Refund Policy
New Subscription/Mail Order Card
Transactor Disks, Back Issues, Microfiche
Sorry, Wrong Number
The MSD DOS Reference Guide
New Services on QuantumLink
PaperClip II for C-128
New Products for C-128 from Abacus
Extend-A-Key
Aegis Art Pak, Volume 1
File Archive Utility for the C-64
FORMATX from Powersoft
Portable Computer Protection
Updated TaxAid
Bookkeeper's Aid for Commodore computers
PROMAL News

TransBloopers . . . 16
Low Cost Universal EPROM Programmer
Keyboard Expander 64: Vol. 7, Issue 3

TeleColumn iNet 2000 and CompuServe data library filename extensions 19
An Introduction to Amiga ML Control the system at the CPU level . . 21
Programming the Amiga It's easier than you think! 24
A Tale of Two Cs A look at C Power and Super C for the 64 and 128 34
Language Speed Comparisons Benchmarks for popular languages . . 39
CP/M block allocation calculator Read CP/M files in 128 mode! . . 42
Evaluating C-128 CP/M Life with the CP/M Beta versions 43
Assembling Assemblers The unique problems of writing an assembler . . . 48
C64 Structured Programming Modernize your ancient BASIC 51
Blazin' Forth An inside look at the Blazin' Forth compiler 58
C128 Programmer's Aid A find, replace, and list-scrolling utility 65
C-64 23K RamDisk A fast way to temporarily save your programs 71
Amiga Dispatches More on the Amiga front 74

Note: Before entering programs, see "Verifizer" on page 4

The Transactor
The Tech/News Journal For Commodore Computers

Editor in Chief
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

D'Artagnan Editor
Nick Sullivan

Art Director
John Mostacci

Administration & Subscriptions
Anne Richard
Kathryn Holloway

Contributing Writers

Ian Adam
Jim Barbarello
Anthony Bertram
Tim Bolbach
Anthony Bryant
Tim Buist
Jim Butterfield
Betty Clay
Joseph Caffrey
Gary Cobb
Tom K. Collopy
Robert V. Davis
Elizabeth Deal
Rolf A. Deininger
Frank E. DiGioia
Paul T. Durrant
Michael J. Erskine
Jack Farrah
William Fossett
Jim Frost
Miklos Garmaszeghy
Martin Goebel
R. James de Graff
Tim Grantham
Adam Herst
John Holttum
David Hook
Tomas Hrbek
Robert Huehn
Tom Hughes
David Jankowski
Bob Jonkman
Brian Junker
Clifton Karnes
Lorne Klassen
Jesse Knight
James E. LaPorte
William Levak
James A. Lisowski
Scott Maclean
David Martin
Steve McCrystal
Stacy McInnis
Steve Michel
Chris Miller
Terry Montgomery
Ralph Morrill
Rick Morris
Michael Mossman
Gerald Neufeld
Noel Nyman
Kevin O'Connor
Richard Perrit
Donald Piven
Terry Pridham
Raymond Quirling
Gary Royal
John W. Ross
David Shiloh
Fred Simon
P. A. Slaymaker
Edward Smeda
Darren J. Spruyt
Aubrey Stanley
David Stidolph
Richard Stringer
Anton Treuenfels
Karel Vander Lugt
Audrys Vilkas
Jack Weaver
Evan Williams
Chris Wong

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725–050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209 **ISSN# 0827-2530.**

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 878 8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') is a straight line as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print ' ' flush right ' ' – would be shown as – print ' '[10 spaces]flush right ' '

Cursor Characters For PET / CBM / VIC / 64

Down – [q]
Up – [Q]
Right – [ ]
Left – [Lft]
RVS – [r]
RVS Off – [R]
Insert – [T]
Delete – [t]
Clear Scrn – [S]
Home – [s]
STOP – [c]

Colour Characters For VIC / 64

Black – [P]
White – [e]
Red – [£]
Cyan – [Cyn]
Purple – [Pur]
Green – [↑]
Blue – [←]
Yellow – [Yel]
Orange – [A]
Brown – [U]
Lt. Red – [V]
Grey 1 – [W]
Grey 2 – [X]
Lt. Green – [Y]
Lt. Blue – [Z]
Grey 3 – [Gr3]

Function Keys For VIC / 64

F1 – [E]
F2 – [I]
F3 – [F]
F4 – [J]
F5 – [G]
F6 – [K]
F7 – [H]
F8 – [L]

Please Note: The Transactor's phone number is: (416) 878-8438

Quantity Orders:

U.S.A. Distributor:
Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)

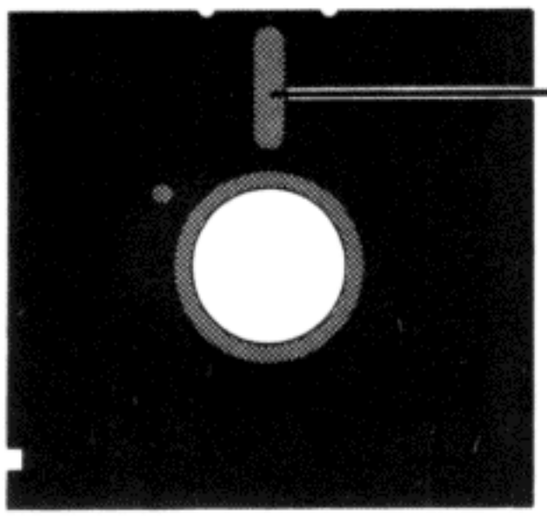
Norland Communications
251 Nipissing Road, Unit 3
Milton, Ontario
L9T 4Z5
416 876 4774

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 04, 05, 06, and Vol 5 Issues 03, 04 are available on microfiche only
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05, 06. Vol. 7: 01, 02, 03, 04, 05

Back Issues: $4.50 each. Order all back issues from Milton HQ.

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos or illustrations will be included with articles depending on quality. Authors submitting diskettes will receive the Transactor Disk for the issue containing their contribution.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Please re-confirm any permissions granted prior to this notice. Solicited material is accepted on an all rights basis only. Write to the Milton address for a writers package.
The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



Start Address

The Amiga is easily the most powerful, flexible and easy–to–use microcomputer available, and is cheaper than anything else in its class. Yet the Amiga doesn't seem to be generating the sales it is capable of (we hear reports, "through the grapevine", of 150,000 units), and we can't help but wonder why.

Part of the blame, of course, must fall with Commodore's marketing, both in their strategy and in amount. Television ads encouraging parents to give their child an unfair advantage by buying an Amiga probably missed the Amiga's market entirely, and the magazine ads weren't much better (what do they mean by 32 instruments?). The people at Commodore–Amiga did and are still doing a great job, but Commodore's marketing department seems to be working against them.

But it's not all Commodore's fault. The stagnation in the micro industry caused by the IBM/MS–DOS standard has made it hard for anyone with a new, exciting product. The first thing you'll hear when you mention a new machine is "how (IBM) compatible is it?" or, "does it run Lotus?". Even for those not of IBM persuasion, there is scorn for any machine that doesn't have a huge base of software available. This brings us to the old catch–22 of no software, no machines sold / no machines sold, no software. But to ensure that a healthy base of good software develops, you have to pick a machine that you believe in and go with it – if enough people do, the software will follow. That's how it worked with the PET, the 64, and every other software–abundant machine. If you buy a machine early in its life, you may have to live with a lack of commercial software for the first year or so, but you have the benefit of having the opportunity at being a pioneer; it's not hard to make new discoveries with a new machine.

A common complaint about the Amiga is its price. This one I can't figure out. The Amiga is much more than just a 68000–based machine with exceptionally fast and flexible graphics hardware, but if even you look at it as only that, its competition is dedicated graphics workstations costing over ten times as much. It's a lot more than a home machine for running games on, but an Amiga system today costs less than a 64 system did at its introduction. Considering inflation, it's considerably less. We all love the 64, but comparing it with the Amiga is like comparing oranges with fruitstands. To be fair, the state of the art has changed, and people are used to paying less for high technology. Compare prices, then, between an Amiga system (512k unit plus monitor) with a Atari 1040ST (with disk drive and monitor). By looking at the machines, you sure wouldn't think they're in the same price range, but they are.

But our purpose here is not to sell Amigas; we have no connections with Commodore, and could write about Ataris just as easily if Commodore passed from the face of the earth. What we are concerned with is the state of the small computer market. It used to be that a machine with more features and power than anything before it would make people line up to get it. Add a fun and easy user interface, multi–tasking, an extensive library of operating system routines, and expandability, and you'd have a hit. That seems to be no longer the case. Has the same market that was attracted to the 64 in '82 evaporated? What, then, of the vertical markets to which the Amiga lends itself: art, music composition, video production? Has Commodore's limited marketing efforts not reached these people yet? Or are they falling on deaf ears? Maybe IBM has made the industry change gears, but the same pioneers that made stars of the PET, Apple II, VIC and 64 could really shift it into overdrive with an Amiga.

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two–letter code for each program line which you can check against the corresponding code in the program listing.

There are five versions of VERIFIZER here; one for PET/CBMs, VIC or C64, Plus 4, C128, and B128. Enter the applicable program and RUN it. If you get a data or checksum error, re–check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 634 to enable the PET/CBM version (off: SYS 637)
SYS 828 to enable the C64/VIC version (off: SYS 831)
SYS 4096 to enable the Plus 4 version (off: SYS 4099)
SYS 3072,1 to enable the C128 version (off: SYS 3072,0)
BANK 15: SYS 1024 for B128 (off: BANK 15: SYS 1027)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing (or "--") it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and usually only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re–check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough; transposing two sets of 4 characters will produce the same report code but this should never happen short of deliberately (verifizer could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking code manually). VERIFIZER ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VIC/C64 VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm–start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI 10 rem* data loader for "verifizer 4.0" *
CF 15 rem pet version
LI 20 cs=0
HC 30 for i=634 to 754:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
OG 60 if cs<>15580 then print "***** data error *****":end
JO 70 rem sys 634
AF 80 end
IN 100 :
ON 1000 data  76, 138,   2, 120, 173, 163,   2, 133, 144
IB 1010 data 173, 164,   2, 133, 145,  88,  96, 120, 165
CK 1020 data 145, 201,   2, 240,  16, 141, 164,   2, 165
EB 1030 data 144, 141, 163,   2, 169, 165, 133, 144, 169
HE 1040 data   2, 133, 145,  88,  96,  85, 228, 165, 217
OI 1050 data 201,  13, 208,  62, 165, 167, 208,  58, 173
JB 1060 data 254,   1, 133, 251, 162,   0, 134, 253, 189
PA 1070 data   0,   2, 168, 201,  32, 240,  15, 230, 253
HE 1080 data 165, 253,  41,   3, 133, 254,  32, 236,   2
EL 1090 data 198, 254,  16, 249, 232, 152, 208, 229, 165
LA 1100 data 251,  41,  15,  24, 105, 193, 141,   0, 128
KI 1110 data 165, 251,  74,  74,  74,  74,  24, 105, 193
EB 1120 data 141,   1, 128, 108, 163,   2, 152,  24, 101
DM 1130 data 251, 133, 251,  96
```

VIC/C64 VERIFIZER

```
KE 10 rem* data loader for "verifizer" *
JF 15 rem vic/64 version
LI 20 cs=0
BE 30 for i=828 to 958:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
FH 60 if cs<>14755 then print "***** data error *****":end
KP 70 rem sys 828
AF 80 end
IN 100 :
EC 1000 data  76,  74,   3, 165, 251, 141,   2,   3, 165
EP 1010 data 252, 141,   3,   3,  96, 173,   3,   3, 201
OC 1020 data   3, 240,  17, 133, 252, 173,   2,   3, 133
MN 1030 data 251, 169,  99, 141,   2,   3, 169,   3, 141
MG 1040 data   3,   3,  96, 173, 254,   1, 133,  89, 162
DM 1050 data   0, 160,   0, 189,   0,   2, 240,  22, 201
CA 1060 data  32, 240,  15, 133,  91, 200, 152,  41,   3
NG 1070 data 133,  90,  32, 183,   3, 198,  90,  16, 249
OK 1080 data 232, 208, 229,  56,  32, 240, 255, 169,  19
AN 1090 data  32, 210, 255, 169,  18,  32, 210, 255, 165
GH 1100 data  89,  41,  15,  24, 105,  97,  32, 210, 255
JC 1110 data 165,  89,  74,  74,  74,  74,  24, 105,  97
EP 1120 data  32, 210, 255, 169, 146,  32, 210, 255,  24
MH 1130 data  32, 240, 255, 108, 251,   0, 165,  91,  24
BH 1140 data 101,  89, 133,  89,  96
```

VIC/64 Double Verifizer — Steven Walley, Sunnymead, CA

When using 'VERIFIZER' with some TVs, the upper left corner of the screen is cut off, hiding the verifizer–displayed codes. DOUBLE VERIFIZER solves that problem by showing the two–letter verifizer code on both the first and second row of the TV screen. Just run the below program once the regular Verifizer is activated.

```
KM 100 for ad=679 to 720:read da:poke ad,da:next ad
BC 110 sys 679: print: print
DI 120 print " double verifizer activated " :new
GD 130 data 120, 169, 180, 141,  20,   3
IN 140 data 169,   2, 141,  21,   3,  88
EN 150 data 96,  162,   0, 189,   0, 216
KG 160 data 157,  40, 216, 232, 224,   2
KO 170 data 208, 245, 162,   0, 189,   0
FM 180 data   4, 157,  40,   4, 232, 224
LP 190 data   2, 208, 245,  76,  49, 234
```

VERIFIZER For Tape Users **Tom Potts, Rowley, MA**

The following modifications to the Verifizer loader will allow VIC and 64 owners with Datasettes to use the Verifizer directly (without the loader). After running the new loader, you'll have a special copy of the Verifizer program which can be loaded from tape without disrupting the program in memory. Make the following additions and changes to the VIC/64 VERIFIZER loader:

```
NB  30   for i=850 to 980: read a: poke i,a
AL  60   if cs<>14821 then print " *****data error***** " : end
IB  70   rem sys850 on, sys853 off
--  80   delete line
--  100  delete line
OC  1000 data  76,  96,   3, 165, 251, 141,   2,   3, 165
MO  1030 data 251, 169, 121, 141,   2,   3, 169,   3, 141
EG  1070 data 133,  90,  32, 205,   3, 198,  90,  16, 247
BD  2000 a$ = " verifizer.sys850[space] "
KH  2010 for i=850 to 980
GL  2020 a$=a$+chr$(peek(i)): next
DC  2030 open 1,1,1,a$: close 1
IP  2040 end
```

Now RUN, pressing PLAY and RECORD when prompted to do so (use a rewound tape for easy future access). To use the special Verifizer that has just been created, first load the program you wish to verify or review into your computer from either tape or disk. Next insert the tape created above and be sure that it is rewound. Then enter in direct mode: OPEN1:CLOSE1. Press PLAY when prompted by the computer, and wait while the special Verifizer loads into the tape buffer. Once loaded, the screen will show FOUND VERIFIZER.SYS850. To activate, enter SYS 850 (not the 828 as in the original program). To de-activate, use SYS 853.

If you are going to use tape to SAVE a program, you must de-activate (SYS 853) since VERIFIZER moves some of the internal pointers used during a SAVE operation. Attempting a SAVE without turning off VERIFIZER first will usually result in a crash. If you wish to use VERIFIZER again after using the tape, you'll have to reload it with the OPEN1:CLOSE1 commands.

Plus 4 VERIFIZER

```
NI 1000 rem * data loader for " verifizer +4 "
PM 1010 rem * commodore plus/4 version
EE 1020 graphic 1: scnclr: graphic 0: rem make room for code
NH 1030 cs=0
JI 1040 for j=4096 to 4216: read x: poke j,x: ch=ch+x: next
AP 1050 if ch<>13146 then print " checksum error " : stop
NP 1060 print " sys 4096: rem to enable "
JC 1070 print " sys 4099: rem to disable "
ID 1080 end
PL 1090 data  76,  14,  16, 165, 211, 141,   2,   3
CA 1100 data 165, 212, 141,   3,   3,  96, 173,   3
OD 1110 data   3, 201,  16, 240,  17, 133, 212, 173
LP 1120 data   2,   3, 133, 211, 169,  39, 141,   2
EK 1130 data   3, 169,  16, 141,   3,   3,  96, 165
DI 1140 data  20, 133, 208, 162,   0, 160,   0, 189
LK 1150 data   0,   2, 201,  48, 144,   7, 201,  58
GJ 1160 data 176,   3, 232, 208, 242, 189,   0,   2
DN 1170 data 240,  22, 201,  32, 240,  15, 133, 210
GJ 1180 data 200, 152,  41,   3, 133, 209,  32, 113
CB 1190 data  16, 198, 209,  16, 249, 232, 208, 229
CB 1200 data 165, 208,  41,  15,  24, 105, 193, 141
PE 1210 data   0,  12, 165, 208,  74,  74,  74,  74
DO 1220 data  24, 105, 193, 141,   1,  12, 108, 211
BA 1230 data   0, 165, 210,  24, 101, 208, 133, 208
BG 1240 data  96
```

C128 VERIFIZER

```
CF 1000 rem * data loader for verifizer 128
HA 1010 rem * commodore c128 - 40 and 80 column mode
DH 1020 cs=0
HL 1030 for j=3072 to 3226: read x: poke j,x: cs=cs+x: next
CB 1040 if cs<>19526 then print " checksum error! " : stop
CP 1050 print " sys 3072,1: rem to enable "
CB 1060 print " sys 3072,0: rem to disable
ME 1070 rem
FG 1080 data 201,   0, 208,  13, 120, 165, 253, 141
FK 1090 data  20,   3, 165, 254, 141,  21,   3,  88
MD 1100 data  96, 120, 173,  21,   3, 201,  12, 240
OJ 1110 data  17, 133, 254, 173,  20,   3, 133, 253
MF 1120 data 169,  44, 141,  20,   3, 169,  12, 141
OM 1130 data  21,   3,  88,  96, 165, 240, 201,  13
EI 1140 data 208,  94, 165,  22, 133, 250, 162,   0
ON 1150 data 160,   0, 189,   0,   2, 201,  48, 144
NH 1160 data   7, 201,  58, 176,   3, 232, 208, 242
IJ 1170 data 189,   0,   2, 240,  22, 201,  32, 240
ML 1180 data  15, 133, 252, 200, 152,  41,   3, 133
DE 1190 data 251,  32, 147,  12, 198, 251,  16, 249
DN 1200 data 232, 208, 229,  56,  32, 240, 255, 169
LM 1210 data  19,  32, 210, 255, 169,  18,  32, 210
LE 1220 data 255, 165, 250,  41,  15,  24, 105, 193
HC 1230 data  32, 210, 255, 165, 250,  74,  74,  74
KE 1240 data  74,  24, 105, 193,  32, 210, 255, 169
OF 1250 data 146,  32, 210, 255,  24,  32, 240, 255
NC 1260 data 108, 253,   0, 165, 252,  24, 101, 250
LF 1270 data 133, 250,  96
```

B128 VERIFIZER **Elizabeth Deal, Malvern, PA**

```
1 rem save " @0:verifizerb128 " ,8
10 rem* data loader for " verifizer b128 " *
20 cs=0
30 bank 15:for i=1024 to 1163:read a:poke i,a
40 cs=cs+a:next i
50 if cs<>16828 then print " ** data error ** " : end
60 rem bank 15: sys 1024
70 end
1000 data  76,  14,   4, 165, 251, 141, 130,   2, 165, 252
1010 data 141, 131,   2,  96, 173, 130,   2, 201,  39, 240
1020 data  17, 133, 251, 173, 131,   2, 133, 252, 169,  39
1030 data 141, 130,   2, 169,   4, 141, 131,   2,  96, 165
1040 data   1,  72, 162,   1, 134,   1, 202, 165,  27, 133
1050 data 233,  32, 118,   4, 234, 177, 136, 240,  22, 201
1060 data  32, 240,  15, 133, 235, 232, 138,  41,   3, 133
1070 data 234,  32, 110,   4, 198, 234,  16, 249, 200, 208
1080 data 230, 165, 233,  41,  15,  24, 105, 193, 141,   0
1090 data 208, 165, 233,  74,  74,  74,  74,  24, 105, 193
1100 data 141,   1, 208,  24, 104, 133,   1, 108, 251,   0
1110 data 165, 235,  24, 101, 233, 133, 233,  96, 165, 136
1120 data 164, 137, 133, 133, 132, 134,  32,  38, 186,  24
1130 data  32,  78, 141, 165, 133,  56, 229, 136, 168,  96
1140 data 170, 170, 170, 170
```

b i t s

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in "Bits", we'll credit you in the column and send you a free one–year's subscription to The Transactor

List 'n' Save

David Mackenzie
Bethesda, MD

I put a line like this at the beginning of nearly every BASIC program I write:

```
5 rem " [6 deletes]open15,8,15, " s0:filename "
  :close15:save " 0:filename " ,8
```

(To create that line, type:

```
5 rem "[6 inserts][6 deletes]open. . .
```

When I " list 5 " , the line number and rem are overwritten by the deletes, leaving just the open, close, and save. Then I just move the cursor up to that line and press RETURN. This technique encourages me to save changes frequently, and I never use the wrong filename by mistake. It also avoids the 1541 save and replace bug (most of the time) by scratching and then saving.

Saving C–128 Function Keys

Daniel M. Bickford
San Francisco, CA

I program my C–128 function keys constantly while I am programming to perform short routines or speed up typing. Of course when I power down I'm back to reset values and, until lately, typing my key definitions back in again.

The C–128 uses memory locations 4096–4351 for programmed keys. Pre–programming my function keys, saving that memory block to disk and having it autoboot I can free myself from typing them in over.

To save the programmed key work area to disk simply:

```
BSAVE " filename " ,U8,B0,P4096 TO P4352
Load with BLOAD or ,8,1.
```

Dvorak Layout and Speedscript

Donald P Maple
Calgary, Alberta

This is for all the people who typed in the *Dvorak Keyboard For The Commodore 64* that appeared in the May 1986 issue and would like to use it from within the *Speedscript* word processor.

The conflict between these two programs occurs in the RAM below Kernal ROM. Both programs use this area, *Speedscript* to store text and the Dvorak program to store a copy of the operating system. The " solution " to all this is to limit the amount of memory available to *Speedscript*. This is unfortunate but the only way short of rewriting *Speedscript* with built–in Dvorak layout.

First of all, type in the following program and save it:

```
10 f = 40832: t = 40866
20 print chr$(147) + " poking from " f " to " t
30 for i = f to t: read x$: print chr$(19)chr$(17)chr$(18);i
40 l = 48: if asc(left$(x$,1))>57 then l = 55
50 r = 48: if asc(right$(x$,1))>57 then r = 55
60 x = (asc(left$(x$,1))–l)*16 + asc(right$(x$,1)–r
   :poke i,x: s = s + x: next
70 if s = 3593 then print " okey–dokey " : end
80 print " something's wrong! " : end
40832 data a9,35,8d,8e,09,8d,98,0e
40840 data 8d,17,16,8d,df,14,8d,e6
40848 data 14,8d,b4,1d,8d,bb,1d,8d
40856 data 02,1e,8d,09,1e,a9,9f,8d
40864 data b1,09,60
```

To implement the Dvorak layout, do the following:
- load and run the Speedscript fix program listed above
- load (but do NOT run) Speedscript
- type " SYS 40832 " and press RETURN

You now have a modified version of *Speedscript*. Save it for future use. All that needs to be done next time is run the

Dvorak program first and then load and run *Speedscript*. Note that the above above modification requires the use of *Speedscript 3.0*.

Finding Relative File Record Lengths

Elizabeth Deal and Howard Harrison

When you open a relative file and specify the record length using the "L" parameter, the file gets built and the record length never changes. But by looking at the file afterwards, do you really know its record length? How about the file you created four years ago? How about a file from some commercial program?

One thing that makes it hard to easily determine the record length of a relative file is that some file are written so that some of their records are shorter than their maximum allowable length (the actual record length specified when the file was created).

The solution to finding the real record length is quite simple: it lies in the good, old, friendly error channel. If you position to byte in a record beyond the record length, you get an OVER-FLOW IN RECORD error. To find the record length, then, you could just code a loop that tried positioning to byte 1 in the record, then 2, etc. (checking the error channel after each position), until the overflow error came up. It couldn't be simpler.

A faster approach is to use a binary–chop technique, as in the program below. This method guarantees that the correct value will be found in no more than 8 tries, rather than a maximum of 255 tries with the sequential method described above. Also, bear in mind that the disk only has to spin once – reading the record over and over again doesn't involve any disk activity, because the contents of the record is stored in a RAM buffer.

The below program will work with any commodore 5 1/4" drive and 8–bit computer.

```
100 rem relfile record–length finder in 8 tries
110 rem elizabeth deal and howard harrison
120 rem   independent of computer and disk roms
130 rem   independent of relfile format (reg or jumbo)
140 rem   does not scan any directory bytes
150 rem   is a read–only routine – uses just channel 15
160 open 15,8,15: open 1,8,3, " relfile,r "
170 mn = 1: mx = 254        :rem record size range
180 if mn>mx then close 1: close 15: print " length = " ;
    mn–1 : end
190 sz = int((mn + mx)/2)   :rem try length sz = midpoint
200 print#15, " p " chr$(3 + 96)chr$(1)chr$(0)chr$(sz)
    :rem rec#1, pos sz
210 input#15,e,e$,e2$,e3$: rem drive knows all about
    length!
220 if e = 0  then mn = sz + 1 : rem valid length, try higher
230 if e = 51 then mx = sz–1 : rem bad length, try lower
240 if e = 0 or e = 51 goto 180
250 print " bad disk error– " e;e$,e2$,e3$
```

VIC Real–Time Clock Fix

Meyer Gottesman Napa, CA

I've discovered if you POKE 37158,147 (the default is 137), you correct most of the realtime clock error. The original IRQ rate was slightly fast. This clock is about 53 seconds in 24 hours fast. With the correction, the error is reduced to about 2.29 seconds. Purists can adjust C–32, the trimmer in series with Y–1 (14.31818 MHz crystal).

Disabling the C–128 RUN key – even more

Ken Smith Milton, Ont.

Having myself once succumbed to the deadly SHIFT–RUN/STOP faux pas, I am quite aware of the value of a fix for this feared bane of all C–128 programmers!

The solution suggested by George Leotti (Bits & Pieces Sep/86) does, as explained, disable the RUN key. There is, however, the unwanted side effect of effectively redefining the HELP key to 'dL" * " + chr$(13)', creating for us the same situation which it sought to alleviate!

The cause of the problem lies in the way the programmable key definitions are stored and accessed. By changing the defined length of the RUN key from 9 characters to 0 (poke 4104,0), without changing the actual length of the definition, the condition for the occurence of the problem is created.

When a programmable key is pressed, the locations $1000–$1009 (4096–4105) are checked for the current definition lengths of the specific key, as well as the lengths of the preceding definitions. The sum of the preceding values is a pointer to the appropriate position within the definition list at $100A–$10FF (4106–4351).

Since the definition length of RUN has been changed to 0 without changing the actual definition, the definition read for the HELP key is in fact the first five characters (the defined length of HELP) of the actual definition of RUN.

Hence, the unwitting CPU happily executes 'dL " * " + chr$(13)' when the HELP key is struck, much to the dismay of the programmer!

Here is my solution. Before altering any of the other programmable keys, modify the definition of the RUN key to the relatively harmless command RUN by poking the following locations:

```
for i=4159 to 4162: poke i,0: next
```

For convenience, combine this modification with any others you may have for making your time at the keyboard more profitable in a short program on the autoboot disk. Below is a partial listing of my "programming mode" file:

```
1 fast
2 print chr$(27);"u": rem underline cursor
3 for i=4159 to 4162: poke i,0: next: rem redefine run
4 key1,"Srun"+chr$(13):rem clear screen and run
5 key2,"dload[16 crsr-right][5 spaces]"+chr$(13)
  : rem load from directory
6 key6,"open4,4:cmd4:list"+chr$(13)+"print#4
  :close4"+chr$(13)
7 (etc. . .)
8 new
```

The following lines are noteworthy:

Line 4 clears the screen before executing a RUN. Saves having to first moves the cursor to a clear spot.
Line 5 loads from a directory listing. Just cursor to desired file and hit F2 key to load it!
Line 6 prints a listing and closes the file afterwards in one keypress.

The suggestions are straightforward, but you'd be surprised at how many folks have yet to set up a small programming aid like this.

C-128 Program Merge — **Don Lawrence, Mississauga, Ontario**

The following procedure will allow you to merge two BASIC programs on the Commodore 128:

```
reset the computer
load the first program
p=peek(174) + peek(175)*256 - 2
poke 45,p-int(p/256)*256
poke 46,int(p/256)
load the second program
poke 45,1: poke 46,28
```

Your computer will now have both BASIC programs in memory as one. The second program will immediately follow the first program even if its line numbers are smaller. Therefore, to avoid confusion, you should make sure that the program's highest line number is less than the second program's lowest line number before merging the two.

You may find it interesting that the pointer to the end of text on the C-128 is not the beginning of BASIC variables as it is on the 64. I found this to be more time consuming than interesting, and I would like to know which fiend at Commodore is responsible for this!

One-Line Direct-Mode File Printer — **Dean Gaudet, Bramalea, Ontario**

Use this statement to dump a sequential text file to the screen from direct mode:

```
open1,8,2,"filename":fori=0to1:sys43906#1,a$
:i=st:?a$;:wait653,1,1:next:close1
```

Use SHIFT to pause the output - the WAIT 653,1,1 halts basic execution if the shift key is pressed.

The SYS 43906# calls the GET# routine, without a check for direct mode.

The 1541 Interleave Factor — **Robert Huehn, Neustadt, Ont.**

While looking for a way to magically speed up my disk drive, I found out how to change the distance between its sectors. The 1541 zero page location at $0069 (105 decimal) holds the "interleave" value, which is normally ten. The interleave is the number of disk sectors between linked sectors in a file; the default value results in a file's consecutive sectors being stored ten sectors apart on the disk. An optimum interleave value means the drive has to wait the least amount of time to read a sector, since the next sector will be close to the head when the drive is ready to read it. To change the interleave value, use the "m-w" command:

```
open 15,8,15
print#15,"m-w"chr$(105)chr$(0)chr$(1)chr$(value)
close 15
```

Any subsequent saves will then be stored on disk with the sectors linked accordingly. (I checked with a sector editor to make sure.) I hoped that Commodore's infinite cleverness had made them overlook a better number than the normal ten. I saved files at different interleaves, each 100 blocks long, and timed the loading speed.

Unfortunately, it didn't have much effect at normal loading speeds. But it made a dramatic effect when I used a fast loader. The same file that took ten seconds at an interleave of eight needed 25 seconds at seven. The reason was obvious: the fast loader has to let those seven sectors pass by while it squirts the previous one through the serial bus. Therefore, if you use a fast loader, you should choose the smallest value that doesn't pass by the head too soon. With the same fast loader, I saved one second by using eight instead of the default ten interleave value.

During a normal load, over 60 sectors may pass while one is being sent by the slow bus routines. Since the gap at the end of the track passes several times, it makes each track and each sector have different optimum interleaves. Anyway, it was a good try.

Fixes For "The Compressor"

John Robert Onda
Lanham, Maryland

"The Compressor" by Chris Zamara, published in Volume 6, Issue 4, Jan. 1986, is a very fine program, and one which seems to have inspired a number of public domain programs. However, in my own experiments with the routine, I have come up against a few problems.

I have worked exclusively with the "Listing 7" version which saves and loads compressed hi-res graphics files to and from memory. I have not experimented with the "Listing 6" version which works on disk files, but similar conditions most likely apply. The following are the three problems I have encountered:

1) The routine reads 8001 bytes, not 8000. "Piclen" is specified as 8000 bytes, but the first byte is read as (byte),y with y=0; subsequently, y is set to 1 and all following bytes are read. This is a very minor and easily corrected problem. Simply set "Piclen" to 7999, or even ignore the situation in most cases (but make note of problem #2).

2) The construction of the "nextout"/"sp1" routines are such that the last group of identical bytes will not be written to disk. (If byte #8001 is different than #8000, no problem is noticed; that is usually the case with the example program "Listing 3".) At the worst this could cause up to 256 bytes of information to be lost from the compressed file. A simple solution is:

```
1210 nextout =    *
1220          jsr   outbyte
1230          lda   picptr+1
1240          cmp endpic+1
1250          bne  nextout
1260          lda   picptr
1270          cmp endpic
1280          bne  nextout
1290          jsr   writerep
1300          rts
[delete lines 1310-1340]
```

3) The variables "repcount", "newbyt", "prevbyt", and "endpic" are specified as ".byte 0"; they are not merely reserved space but must be zeros upon entering the routine. If left at previous values and the routine is called again, results are unpredictable. A simple solution is to clear these bytes before entering the main routine (jsr clear before entering the compressor):

```
clear  lda #0
       sta repcount
       sta newbyt
       sta prevbyt
       sta endpic
       sta endpic+1
       inc repcount
       rts
```

Also, "Listing 5", which splits Animation Station files, incorrectly assumes the background colour is stored as the last byte of the file. In fact, the background colour is stored at a byte within the "extra useless bytes" skipped in line 310. In addition, Animation Station stores a border colour byte preceding the background colour bytes. One solution:

```
310 for a=1 to 63: rem useless bytes
311 get#8,a$
312 next
313 get#8,a$: bo=asc(a$+chr$(0)): rem border colour
314 get#8,a$: ba=asc(a$+chr$(0)): rem background
315 for a=1 to 127: rem useless bytes
316 get#8,a$
317 next
[delete line 410]
440 print ba
441 print"qq The border colour is:"
442 print bo
```

It is also worth noting that the "Animation Station" software is a custom version of the program "Blazing Paddles" by Baudville and the file formats are identical.

For those working with Koala files, which do not save a separate border colour byte, generally the background colour is used as the border colour; replace in Listing 4:

```
400 print "qq The background and border colour is:";
```

(Thanks for your efforts in correcting the compressor, John! The bugs were fixed in later re-incarnations of the compressor code, but we never printed the fixes or a de-bugged version. Fortunately, because of people like you helping us out, our readers are kept up to date! –CZ)

C-64 Time of Day Clocks

Raymond F. Genovese
Richmond, VA

The two time-of-day (TOD) clocks in the 6526 CIA chips are often overlooked. They keep time in 0.1 second increments and are not disturbed by either tape or disk I/O. In fact, TOD clock #2 is not used at all by the C-64's operating system. Starting at locations $DC08-$DC0B (CIA #1) and $DD08-$DD0B (CIA #2) their registers are as follows: tenths (bits 0-3), seconds (bits 0-3 & bits 4-6), minutes (bits 0-3 & bits 4-6),

hours (bits 0–3 & bit 4). The digits are loaded and stored in binary coded decimal format, and thus are a little awkward to work with. Writing to the hours register stops timing while writing to the tenths register starts it. On the other hand, reading from the hours register latches the time (but does not stop it) while reading from the tenths register unlatches the time. Listing 1 is an example routine that sets, reads, and displays a TOD clock. Features not implemented in this example are the AM/PM flag (bit 7 of the hours register) and the interrupt alarm features. In addition to keeping time, these clocks can be used to control fairly precise time–delay loops. Listing 2 demonstrates that technique.

Listing #1

```
MO  100 rem tod clock example
FK  110 gosub 150: print chr$(147)
MG  120 gosub 260: print t$;chr$(19)
PB  130 goto 120
AA  140 :
OJ  150 rem set and start the clock
LA  160 ba=56331: rem cia#1
OM  170 input" enter the time (hhmmss) ";t$
DA  180 if len(t$)<>6 then 170
NH  190 for x=0 to 5 step 2
AE  200 y=val(mid$(t$,x+1,2))
JD  210 y=int(y/10)*16+(y-int(y/10)*10)
KO  220 poke ba-x/2,y: next x
DD  230 poke ba-3,0
MA  240 return
OG  250 :
CF  260 rem read the time
AG  270 t$="":m$=" am":for x=0 to 5 step 2
MF  280 t2$=str$((peek(ba-x/2)and 112)/16)
        + str$(peek(ba-x/2)and 15)
AK  290 t$=t$+mid$(t2$,2,1)+mid$(t2$,4,1)
MK  300 if x<>4 then t$=t$+":"
GI  310 next x: x=peek(ba-3)
MF  320 return
```

Listing #2

```
OC  100 rem 5-second delay using tod clock
KP  110 ba=56584: rem cia#2
GE  120 poke ba+3,0: poke ba+1,0
JM  130 poke ba,0: rem start clock
GG  140 if (peek(ba+1)and 15)<5 then 140
PF  150 print" time's up! "
AK  160 end
```

Screensave for the C64

Steve Lukshides
Philadelphia, PA

Screensave gives you peace of mind – you can leave your 64 unattended for as long as you like, and not worry about burning out the monitor CRT!

This short program for the C–64 sets up an interrupt routine that will help prevent images from being permanently burned into your monitor screen. This can happen if the screen stays on for a long period of time while displaying the same image – you can observe this phenomenon by looking at the screens of older video games at an arcade. Screensave will blank the 64's screen if there is no keyboard activity for two minutes – pressing any key (including shift, CTRL and logo) will turn the screen on again, with its original contents intact. Screensave runs in the background while it monitors the keyboard for activity, so it doesn't affect the normal operation of your system when it's active.

Just run the loader below to put screensave into the cassette buffer and activate it. Screensave is relocatable, so you can change lines 20 and 60 to store and execute it in any memory area you wish. To disable screensave, use SYS 864.

```
CK  10 rem* data loader for "screensave" *
OB  20 for i=828 to 1003:read a:poke i,a
JG  30 cs=cs+a:next i
MJ  40 :
LA  50 if cs<>19715 then print"!data error!":end
AC  60 sys 828
HP  70 print" screen saver activated!"
AF  80 end
OM  90 :
LM  1000 data 120, 173,  20,   3, 141, 236,   3, 173
GO  1010 data  21,   3, 141, 237,   3, 169, 111, 141
KB  1020 data  20,   3, 169,   3, 141,  21,   3, 169
NN  1030 data   0, 141, 240,   3, 141, 241,   3, 141
FF  1040 data 238,   3,  88,  96, 120, 173, 236,   3
HC  1050 data 141,  20,   3, 173, 237,   3, 141,  21
ME  1060 data   3,  88,  96,  72,   8, 169,  16,  44
NM  1070 data  17, 208, 240,  41, 169,  64, 197, 203
GG  1080 data 208,  60,  32, 197,   3, 173, 238,   3
HF  1090 data 240,  22, 173,  32, 208, 141, 239,   3
KA  1100 data 169,   0, 141,  32, 208, 141, 238,   3
PO  1110 data 169, 239,  45,  17, 208, 141,  17, 208
HM  1120 data  40, 104, 108, 236,   3, 169,  64, 197
IF  1130 data 203, 208,   5, 173, 141,   2, 240, 240
NH  1140 data 173, 239,   3, 141,  32, 208, 169,  16
LE  1150 data  13,  17, 208, 141,  17, 208, 169,   0
HH  1160 data 141, 240,   3, 141, 241,   3,  76, 156
BB  1170 data   3,  24, 169,   1, 109, 240,   3, 141
CK  1180 data 240,   3, 144,  10,  24, 169,   1, 109
HO  1190 data 241,   3, 141, 241,   3,  96, 169,  32
CK  1200 data 205, 240,   3, 208,  10, 169,  28, 205
PD  1210 data 241,   3, 208,   3, 238, 238,   3,  96
```

Letters

Copy-All 64 and relative files: A version of Copy-All 64, which is on the games disk (Transactor disk #13) and another early disk, cannot copy relative files. The PET version is all right. Every once in a while your readers ask how to copy, device to device, relative files. For instance, on page 15 in Volume 7 Issue 2 was a query about an SFD to SFD copy. It seems to me that Copy-All should handle that with little effort. But there is a general misconception that Copy-All can't handle relative files. Well, this isn't true. Jim Butterfield, long ago, fixed up the C64 version, but the T must have a unique version.

The culprit is the CLRCHN routine which, unlike on the PET, in the C64 clobbers the X register (as well as .A). Now, there is a little piece of code in the old relative file section that goes like this:

```
JSR  $FFE4  ;read first byte of ch. 15
TAX
JSR  $FFCC
CPX  #$30   ;is it "0"?
BNE  QUIT   ;quit if not
LDA  $90    ;read ST, loop if good
etc.
```

The result is that the $FFCC routine immediately makes X not equal to "0" and causes Copy-All 64 to quit prematurely.

The solution is simpler than you think. $FFCC does not touch the Y register. So substitute TAX by TAY and CPX by CPY, and you're in business. The relative file copy code begins at $116E and the bytes to change are a screenful later. You can make the changes using the monitor and re-save.

Elizabeth Deal, Malvern, Pennsylvania

Thanks for the correction, Liz. Copy-All 64 goes on every Transactor disk, so it's kind of embarrassing that this little bug has got past us for so long. However, all future T. disks will go out with the corrected version of the program, and Copy-All, one of the most useful and durable of all Commodore programs, will be even better than before.

Adapting Search and Print: Mr. Boland's letter in the September 1986 issue was an excellent update on the various enhancement packages available for SpeedScript, which gets better every day! My article (Four wordprocessors) was submitted on October 10, 1985, at which time none of the enhancement modules were available, hence their absence. There is usually a delay of several months between submitting an article and its publication and a lot can happen in that period as was true in the case of SpeedScript.

I would like to take this opportunity to congratulate you for doing such a great job. Transactor is a unique magazine and gets better as it grows. I would also like to convey my praises to Jack R. Farrah for his Hi-Res Search and Print utility which is a godsend! Users of B/Graph would find it extremely useful, especially the double width screen dump. This can be easily fiex by changing the '40' in line 1310 to '41'. This would fix "clmcnt" so that 40 columns are printed before the "ckclm" routine quits.

Also, those who have Epson-code compatible printers (Epson, Panasonic, Roland etc.) should change the '51' in line 1290 to '65' and the '16' in line 1300 to '8' and they should be able to use the program as well. If Mr. Farrah is reading this and is thinking of further upgrading his program, he might like to enchance the density of the printout by using the high density feature of these printers (120 dots per inch). This makes a nice dense printout which can be photographed for slides or photocopied for overhead transparencies.

I hope the above suggestions help.

Ranjan Bose, Winnipeg, Manitoba

Converting to Merlin: I have recently subscribed to the Transactor and am interested in the programs listed as source code. My assembler is Merlin. Do you have any instruction sheets that would allow a novice to convert the programs as listed to be used with my assembler?

Any help along these lines is appreciated.

Edward F. Weller, Jr., Sun City, Arizona

In most cases, translating between assembler formats is not that difficult a job. Most assemblers adhere closely to the standard created for the 6502 chip when it was first brought out; this is true of both PAL (the assembler we usually use) and Merlin.

There are still a few things to watch out for. The main source of incompatibility is the 'pseudo-ops', like .BYTE and .WORD, which can vary pretty wildly from one assembler to another. PAL's repertoire of pseudo-ops is not large, and in nearly all cases you should be able to find ready equivalents in Merlin. The three most commonly used are .BYTE, .WORD and .ASC, which allow you to set up tables of bytes, 16-bit words and (Petscii) text respectively.

At the top of PAL programs you'll generally see a directive beginning with the .OPT pseudo-op, which specifies where the object code output is to be sent. The usual possibilities are .OPT OO, which assembles to directly to the origin address, and .OPT On, where n is the number of an open disk file. The origin itself is set with a line like:

```
100 * = $C000
```

instead of the .ORG directive used by most other assemblers.

Occasionally you'll see source listings that are so closely tied to the special capabilities of one assembler that they are very difficult to adapt to a different one. An example is the Help utility in this issue, which makes use of PAL's .BAS pseudo-op. Since .BAS is not supported by any other assembler we know of, you would have to do some serious translation to port the source of this particular program.

'With permission from the T.': I wish to copy one or two short articles (not more than a page, and no long listings), from various issues of the Transactor for inclusion in our computer club newsletter

– the "Pasadena Commodore Computer Club 'Tid Bits'", which is distributed free to members, giving credit to you and to the authors.

I am not a club officer, just one of its approximately 400 members. I enjoy your magazine so much that I think the other club members would enjoy utilizing the information also.

I note that J. Butterfield includes a release at the beginning of each of his articles; this would of course be included with any of his work.

For articles by other authors, including the Transactor staff, is it necessary to get a copyright release for each article individually, from you – and/or from the author, or is it sufficient to give credit as noted above in this free newsletter. If an article by article release is required, I may just skip the whole thing, because the clearance procedure is so time–consuming, thereby depriving the many members of your expertise.

I have been a subscriber to the Transactor, both the magazine and the disk, for several years and also have your "Best of" series. I plan to continue the subscriptions, at least for the foreseeable future. Keep up the same quality of content and I predict a long and successful future.

George I. Taylor, Jr., La Canada, California

Well, George, ahem. . . if ensuring that the members of your club aren't deprived of Transactor material is the main concern here, then I have an ideal solution which need not even be told. That "solution" times 400 would also contribute towards our "successful future" much more than the quality of our product. However, re-printing one or two particular articles (and this goes for all User Groups) is perfectly OK, providing the author and The Transactor are credited.

Plus/4's in our future? I have a Commodore Plus/4 computer. Please advise if your magazine will continue to support the Plus/4 in the future.

Mrs. Kathe L. Holiday, Cape Coral, Florida

As you might expect, we are not exactly flooded with submissions relating to the Plus/4, nor do Plus/4 users constitute a very large portion of our readership. That means that you probably won't be seeing a lot of Plus/4 material in issues of the T. from this point on. Probably your best bet is to get in touch with PLUG, a Plus/4 users group. You'll find the address, and a lot of other information, in the next letter, which Jim Butterfield extracted for us from his voluminous correspondence.

Plus/4 anguish, addressed to Jim Butterfield: I am a Commodore Plus/4 computer owner. Since both the +4 and C16 are orphan machines, I've had to write most of my own software because there just isn't any viable support.

In the June issue of the Transactor, I read with interest your C128 memory maps. The Plus/4 runs a 3.5 subset of the C128 7.0 BASIC and the two machines are very compatible in some respects. If I ever get upgraded to a C128 I'm sure I'll have little difficulty translating +4 map locations to C128 thanks to the excellent article.

I belong to a local Commodore users group where members mainly use C64 and C128. Mine is the oddball machine in the lot and I have to convert most of the good C64 software I find. One of the first things I did after joining was to check out from the library your book on C64 machine language programming. Excellent book! You're a good writer and I learned a lot about C64 internals. My big problem is the conversion of software from the C64 map to the +4 map.

BASIC conversions are not much problem for me if they contain just the most common PEEKs and POKEs. There is only one modem (1660 with modified +4 Higgy–Term software) that I'm aware of that will run on the +4. Since I already have a C64 Total Communications Package 300 baud modem I've been trying to make it work with the +4. Big problem is that I can't convert all of the C64 PEEK & POKE addresses to the +4 equivalents. What I'm needing are some very detailed memory maps (all of memory including Kernal and BASIC) for both C64 and Plus/4 machines so that I can make the conversions and see if the modem will work. When it comes to ML in DATA statements I'm totally lost because I've not yet written the translator software to do table look–up and replace on the C–64 –> Plus/4 address conversion.

First of all, Jim, can you help me find the memory maps I need to set up the conversion tables between the two machines?

Secondly, Jim, can you tell me how to write or find a piece of software that I can modify (maybe a kind of pre–processor that will make language translations) to take a lot of the conversion drudgery out by doing a look–up on the conversion tables and then writing and outputting the translated Plus/4 code?

'DEARTH' is one of my favourite words, and there is a pretty complete DEARTH of Plus/4 support. I've got dozens of pent–up questions to ask someone in the know like yourself, but I'll spare you the pain and just hit some of the biggies.

For your information Jim:

PLUG (The PLUS/4 Users Group)
Box 1001
Monterey, California 93942

is one of the most active support channels for the +4 computer. I am a member even though it's half–way across the country from where I'm at here in Arkansas. Calvin Demmon (a writer like yourself) is PLUG editor, and if you have any 'pull' with Commodore, or could give PLUG a plug in your upcoming books or articles, it sure would help generate a little more interest and support.

More Plus/4 computers have been sold than are actively in use, and most owners probably gave up on it because of Commodore's lack of marketing and software support. If we can get out the word that there is at least one active Plus/4 support avenue here in the USA, maybe the support will begin to coalesce and suppliers will take notice.

Also Jim, I would like to get in more assembly–level programming on the +4. Plus/4 has a built–in monitor similar to C128 and the C64 Supermon. However, I very much need a true assembler. Have you seen any assembler software (however slow) that is written in BASIC or ML that I can easily convert and implement on the Plus/4?

In a past issue of COMPUTE!'s Gazette, there was an article entitled 'VIC Emulator' which ran on a C64 to emulate the VIC computer. It's been in the back of mind to try and take some of the emulation techniques used to write a transparent emulator wedge (better yet a pre–processor) that would run on the +4 and allow use of the vast majority of C64 software that is available. Seems to me that such a compiler or language translator writer might just accomplish such a

task if the hardware will allow. I am neither, and don't have the talent or resources to tackle this kind of project but if you know of anyone who is there are a lot of software hungry +4 owners out there who would love to see such a piece of software (maybe even on a commercial basis if the marketing effort is right). From my limited knowledge, writing a piece of software like this probably wouldn't be a very easy task without the proper resources and backing, but then I'm not in a position to say so definitely or even if the potential risks are too great because the +4 is an orphan machine.

Gary Hearn, N. Little Rock, Arkansas

Jim's reply: *Dear Gary,*

The Plus/4 is a fine machine, but is indeed an orphan. You seem to be coping well. The following may be of interest.

–The Inner Space Anthology, published by the Transactor, contains detailed maps of RAM and ROM for the Plus/4 and C16. I've tried to keep wording consistent between maps, so you can do comparisons with the 64 or 128 as needed.

–My book does cover the Plus/4 and C16, and you can run almost all the exercises on those machines. Don't let the title mislead you into thinking that it's only for the 64 or 128.

–Many BASIC programs will run with little or no conversion. Snoop out the POKE and PEEK statements to see if conversion is needed. The greatest work is in the area of sound and graphics, since a completely different chip is used.

–Machine Language may convert fairly well. Much of the problem you'll need to solve in doing a conversion is to allow for a new program location. Commodore 64 BASIC normally starts at address $0400, 128 BASIC at $1C00; on the Plus/4 and C16, you'll normally start at $1000. Thus, the machine language program will also likely move up or down the same distance. Often, this isn't hard to fix: disassemble the code and look for three–byte instructions; the third byte is all that will need changing. As in BASIC, there are other locations that may need adjusting.

–Commercial software is almost hopeless, especially if it has a protection scheme. Such programs are too massive for an easy rechop.

–I don't know of any conversion/emulator programs that would be useful for your problem.

–I'm in touch with PLUG, and have donated some utility programs to them. It's very useful for orphan–owners to have such a group.

–The SYMASS assembler, published by the Transactor, is public domain and source is available. It's likely your best bet for conversion to Plus/4.

Jim Butterfield, Toronto, Ontario

ML column bandwagon grows: Please let me be the 'nth' to join in Rick Nash's timely plea for a Machine language Column. Let us hope that your mail bags all burst in the generous flood of responses to this important suggestion.

There must be countable thousands (or possibly even uncountable) "out there" who are nearly rank–quality amateurs and who would value such a column beyond all reasonable 'a priori' expectations.

Being in agreement with Nash, however, does not mean that I feel restricted to his recipe for success. My own thoughts are quite different and, I sincerely hope, so are those of other respondents.

My own deep desires for an ML Column include:

1. Large numbers of small ML programs that *work!*
2. Clear reference to the publications used including page numbers.
3. Documentation of each line with reasons why choices were made.
4. Author information, including address, response category (does he or doesn't he)?
5. Does author require SASE (or is he independently wealthy)?
6. What assembler was used.
7. Mention of reference(s) pertinent to the subject routine.

Doubtless there are many other facets to this intriguing suggestion which will be forthcoming from other inspired readers. I, for one, truly hope so.

Finally, in your quest for new readers, how could you do better than to extend a bi–monthly helping hand (the Transactor Machine Language Amateur's Column) – possibly, page – ultimately book!! – to these sincere and deserving people?

Robert G. Tischer, Starkville, Mississippi

Although a regular tutorial column on machine language for novices isn't totally out of the question, we do have a couple of reservations about the idea. For one thing, you can't hope to learn machine language in a series of small bimonthly doses - it would just take too long. In our opinion, you'd do a lot better to learn the basics from a book (Jim Butterfield's would be ideal) then, once you're over the initial hurdle, glean what you can from the source listings we publish in every issue. Sure, there'll be things you don't understand at first, but you'll be surprised at the proficiency you can attain by banging your head on code that's too advanced to follow completely. And the other thing you have to do, of course, is write programs - lots of them, until it starts to come naturally. The hardest part of learning machine language is the painful process of getting familiar with all those instructions and addressing modes, and mastering some simple sequences to do common operations like 16-bit math, setting and clearing bits, and moving chunks of memory around. Once you're through that phase, the going starts to get a lot easier.

In search of PET classics: I have just recently heard of the Transactor, and I must say that I am most impressed by it. I particularly like the small amount of advertising, and the large amount of utility–type programs.

I wonder if it would be a practical proposition for you to publish some of the old, but good, public domain programs for the PET/CBM machines, or possibly instruction on converting C/64 programs to CBM.

You see, it is quite possible that there are people out there who, like myself, purchased a second–hand CBM at a time when those machines were cheaper than a new C–64. I would think that there are a lot of newcomers to the world of computing who would like to get hold of some of the old PET classics. I know that this type of request is usually met with "Join a user group". Well, I did that, but no good public domain stuff for PET.

Also, I have now received my G–link and have installed it as per instructions. The G–link works in both serial and parallel (IEEE)

mode. However, the G–link will not switch from serial to IEEE and back again any more than two times before the computer locks up and has to be reset (or switched off and on).

I am using a C–64, 1541 drive and a SFD 1001. The idea is to have the G–link switched to serial, load a program from the 1541, switch to IEEE and save the program to the SFD 1001. This can be done twice and the C–64 locks up; sometimes just the C–64 locks up, sometimes the cursor just disappears, other times it remains immobile on the screen. Now maybe it's not designed to go back and forth between serial and IEEE in the manner in which I am using it. If that is the case then I have a bonus; if it *is* supposed to switch back and forth between serial and IEEE without locking up the C–64, then it would appear that something is wrong. I would like to stress that it will work the SFD–1001 on its own, as it will the 1541 on its own. Anyway, the thing is a bit of a puzzle to me. Could you give me some more info? Does it work with the C–128?

Anyway, I still think that the G–link is pretty good, but I want you to know what is happening – just in case Murphy's law is operating (is it?).

Bill Bennett, North Lidcombe, New South Wales, Australia

Sorry, Bill, but it just wouldn't be practical for us to start rerunning PET listings in the Transactor. As it stands, we already don't have room for all the material we would like to run on the ***new*** *computers, and there just isn't the demand to justify reprising PET oldies. We do still run PET versions of some of our new programs, when possible, so don't think we've given up on the green screens altogether. Also, the first couple of Transactor disks have lots of material for the PETs, including some of the classic utilities that you'll use again and again.*

Your G-Link is behaving normally. Switching back and forth between serial and IEEE modes can, and frequently does, crash the computer with the symptoms you describe. As far as I know, nobody has come up with a definitive explanation of the cause of this crash.

Luckily, though, there is a way around it. It appears that the crash only happens when the cursor is on the screen. That being the case, all you have to do is ensure that the cursor isn't active when you flick the G-Link switch. If you don't use a Datasette, the easiest way of doing this is to press SHIFT-RUN/STOP, flick the switch, and then press RUN/STOP (unshifted). Otherwise, you could type WAIT 653,1, flick the switch, and then press the SHIFT key.

As for using the G-Link with the C-128 - sorry, it won't work, except in C-64 mode. The G-Link operates by replacing the C-64 Kernal ROM with a ROM of its own, and that isn't going to help you at all in C-128 mode. If anybody does know of a good IEEE interface for the C-128, we'd appreciate it if you'd let us know so that we can pass it on.

More GAMES feedback: I totally agree with Wayne Gurley, (Letters, November), relative to the September issue on Games.

I have designed ML utility programs and a software protection scheme. Now I am trying my hand at composing my own arcade game which employs graphics. I have searched in vain for a book that contains advanced techniques like raster interrupt and horizontal zone scrolling. I wrote to a number of software producers with zero response. It would have been nice if the Games issue contained some techniques used in arcade games. There are tons of 'Beginner' books, but absolutely nothing, to the best of my knowledge, for advanced.

John Augustine, Reading, Pennsylvania

Okay, everybody who wants to see another GAMES issue, with the emphasis on graphics and sound, put up your hand. Or at least send us a postcard, telling us what it was you would have liked to see in that issue, but didn't. If the response is sufficient, maybe we'll take another crack at games in a future issue. Meanwhile, if you take a look back through past issues as recently as Vol. 7, Issue 03, you'll find a small article on raster interrupts that will help you get started with your project.

Unassembler and Symass fixes: I am writing to thank you for two wonderful programs. I am one of those who entered Symass from the source listing and have converted it to write the ML program to disk as a program file. I also added the Ascii codes for 'end', 'or', 'eor' and 'ror' so that I can assemble source code which has been written directly to disk.

This last modification was needed because I also converted Unassembler to write directly to disk as a program file. Rather than send the token for each BASIC keyword, I chose to send the Ascii for the keyword.

During the process of changing both of these fine programs, I discovered several problems with them. Symass has only one that I am aware of. The listing on line 1020 should be CMP #"." rather than CMP #" ". Symass has changed my life!

Unassembler has a number of problems. The most severe one is associated with the 'absolute' commands. The reported problem with the 'bit' command (A BIT of a Problem, May 1986) is a result of this defect. It is not solved by simply changing all the bit commands to .byte commands.

The problem shows up when there is a bit command in regular code and also when there is any absolute mode code which has an address less than 256. This will occur when the Unassembler is in Ascii tables and encounters the following sequence: 'absolute command, address, .byte 0'. The Unassembler recognizes the command. It then takes the next two bytes and treats them as low/high. Multiplying the second byte by 256 results in zero. It then stores the result as 'zero page command, address', dropping the final '.byte 0'.

To correct this problem, load Unassembler and enter the following lines:

```
1380 pp$ = "  ":ad$ = "  ":if t = 0 then 1405
1405 if n = 14 then 1430
1410 if n>0 and n<14 then pp$ = ad$ + n$ + "  "
1430 if n>10 then on (n–10) gosub 1950,1980,2010,2015
1445 if n = 14 then 1380
1732 if ad<256 then n = 14:return
1792 if ad<256 then n = 14:return
1852 if ad<256 then n = 14:return
2015 rem ***convert absolute address less than 256 to
     .byte commands
2020 p$ = ad$ + " .byte " + str$(op) + " :.byte "
     + str$(ad) + " :.byte 0   ;***was " + n$ + " *** "
2022 n = 0:gosub 2150:p = p + 2:return
```

This modification will substitute .byte commands for the offending absolute command and notify you what command was changed.

Also, the proper way to implement the 'bit to .byte' conversion is:

```
330 print " Do you want bit commands converted into .byte
    commands? "
340 get an$:if an$ = " " then 340
350 if an$<>"y" and an$<>"n" then 340
360 if an$ = "y" then md(36) = 0:md(44) = 0
362 if an$ = "n" then md(36) = 14:md(44) = 14
```

Another minor problem with Unassembler when used with Symass is in line 2010. The + " A " has to be removed or Symass will assign a label to the 'A'.

By the way, one of the best ways to test both Unassembler and Symass is to unassemble a short program such as C–64 wedge. Reassemble it with Symass and it will be exactly the same if both programs are working correctly. I have tested it on several programs and it always works.

Thomas W. Gurley, Wills Point, Texas

P.S. I am the only staff programmer for the Pondaroda Nudist Resort. It sure is a hard life!

Thanks for the detective work, Thomas. And you certainly have our sympathies over your work situation. Let's hope things will get better soon. It's a sobering thought for us, freezing complacently here in Canada, that others, less fortunate than ourselves, are forced by a cruel fate to run around with no clothes on in the Texas sunshine and program microcomputers into the bargain. Who can understand the workings of Fate?

Symass POKE discrepancy: Problem: in Vol. 7, Issue 2, Page 40, you published a bug correction (Symass 3.12).

In Vol. 7, Issue 3, Page 14, you did another (Symass 3.13). But the third pokes in these two articles do not agree. Which is correct?

Emil J. Volcheck, Jr., West Chester, Pennsylvania

Very observant, Emil! For the benefit of those who don't happen to have those issues handy, the pokes in question are: POKE 5057,50 (Issue 2), and POKE 5057,51 (Issue 3). We're happy to report this isn't a typo. What the second poke corrects is simply the version number (3.13 instead of 3.12). The 50 in the first poke is the Ascii code for the numeral '2'; the 51 in the second poke is the code for '3'.

1541 upgrade ROMs: In the November 1986 issue on page 77 there is an offer for ROM upgrades for the 1541 disk drive selling for $49.95.

What I am not sure of from the information in this column is whether the two ROMs are pin for pin compatible with the Commodore ROMs (direct replacement), or are the provided ROMs of the type that would need the 28 to 24 pin adapters?

Terrence Smith, Lachine, Quebec

The ROMs will plug right in, Terrence. As you're probably aware, the 24-pin EPROMs are less common, hence more expensive, than the 28-pin type. We could have used the 28 pin EPROMs, but this would have required 28 to 24 pin adapters for each chip and the price would have been no less. Besides that, the adapters raise the chips so high that you can't get the lid back on the disk drive once they're installed.

Super Kit - the dark side: I finally received the Super Kit 1541 after a long wait. Was it worth it? I don't think so. The Super Kit turned out to be a Super Disappointment.

After slaving the whole weekend over this thing, I still have to produce a workable copy of the Super Kit as instructed in the accompanying booklet. I checked out my drive with an alignment program, then one that reads track 1 and 35 only, and finally another one from the original 1541 demo disk. My drive is in alignment. After that I checked my drive speed and that seems to be in order too. My drive was checked out 6 different ways and there is nothing wrong with it.

Single Nibbler: produces a non–working copy of a disk with one single error 23 on and places it elsewhere. I duplicated one disk that had an error 23 on track 35, sector 16. It reproduced the error faithfully on track 35, sector 3. Ergo, my program won't run.

Single Copier: Produces a workable copy of an unprotected disk. However, only in the non–verify version. The Single Nibbler does the same. It reads the first pass, then proceeds to write the first pass. Halfway through writing the first pass, the drive does a double–take, continues for a while, then stops, as it should, waiting for the second pass. Meanwhile, the screen stays bright, there is no message for the second pass, and the computer is locked up and nothing works.

Scan: D option sends drive off into never–never–land most of the time.

Disk Editor: In monitor mode gives wrong conversion: when asked to convert 2049 decimal to hex, the answer is not correct, it comes up with $0701. Binary is also wrong. Same thing for 1024 decimal: it converts it to $0300. When asked the equivalents for $801 it brings the right answer. The decimal to hex and binary is wrong.

Disk Surgeon: I was able to produce one copy of Karateka that was actually working. I tried Skyfox. It produced a copy that at the end of the loading process made my drive 'sing'. I thought it had stripped some gears and I had a hell of a time getting the drive initialized.

With the Super Nibbler I came very close to producing a working copy of Super Kit. However, the flashing border was absent and it showed the first page of the menu, and locked up the computer.

Meanwhile, my expensive programs are still as vulnerable as ever, without having a back–up copy.

I have tried to duplicate Super Kit at 38, 39 and 40 tracks to no avail. My system is a C–64, a 1541, and a 1702 monitor. The joysticks were disconnected. There was nothing else on the system, as I do not own any other peripherals.

According to your note for Super Kit owners on page 13 of the Transactor, Vol. 7, Issue 3, November 1986, there seems to be a rewrite under way at Prism, to do away with these kinds of problems.

I would appreciate your suggestions on this problem. Do you accept a return of the product, as it is of no use to me at this stage, or should I wait for a trade–in at a later stage (if you intend to make trade–ups to the latest version)?

Roger Detaille, Ste. Therese, Quebec

Your problems with Super Kit illustrates some of the difficulties that seem inevitably to attend this type of software. Super Kit is fast -

blindingly fast - at both conventional and unconventional DOS operations, and its error copying and scanning are among the most sophisticated you'll find anywhere. Unfortunately, there are some jobs that seem to work reliably only on the most perfectly set-up 1541, and copying SuperKit itself (for which you should use the Super Nibbler module, by the way) is one of them.

While the newer 2.0 version of the program does away with many of the problems you mention in the other modules, Prism Software's president, James Domengeaux, admits that many users have experienced difficulty in backing up SuperKit as advised in the manual. Prism is currently working on ways around this problem, and we're hoping to see even newer updates fairly soon.

Meanwhile, if you have the 1.0 version and want to upgrade to 2.0, we don't think you'll be disappointed with the quality of Prism's after-sales support to registered users. The 2.0 Super Kit is even more powerful and versatile than the first, and seems also to be more reliable. Again, though, you can only expect consistently good results if your drive timing and alignment are fairly accurate.

Squashing C-64 RS232 bugs: What prompted me to write was the exchanges on RS232 in Vol. 6, Issue 6, Page 11, and a point not covered in Lyle Giese's follow-up in Vol. 7, Issue 2. Albert Harsch cites 'buffer problems' at 1200 and 2400 baud. Examination of the Kernal at $EF39 onwards reveals the main RS232 bug in the C64: if DSR or CTS disappear, the NMI interrupts are disabled. When these signals return to 'OK', the NMIs *are not restored* except by a PRINT#2 physically executed after these lines return. As a result, x-line output will leave the last buffer untransmitted unless:

a) the code is altered (my preference)
b) (up to) 256 Ascii nulls are PRINT#2-ed
c) after a real-time delay, some PRINT#2s are issued

As the last two are inelegant, and I use an EPROM Kernal, I chose the first. This requires modification to both the output code at $EF39 and the CLOSE routine. Incidentally, while I was remodelling in this area, the buffers were shifted to pages $CE,CF. This permits OPENing an RS232 channel at any time in a program, without a CLR. (Although pages $DE,DF would have been preferable, they do not work in this context.) This will facilitate use of serial printers with a variety of utilities such as Easy Mail.

After fixing the 75 baud error in the PAL table, I also implemented 3600 and 4800 baud without any problems with printers. (Incidentally, the theoretical maximum baud rate for a PAL C64 is about 5000). Final testing showed, however, that Easy Script and my Kernal did not get along if an RS232 printer was tried. As I have 3 printers, all RS232, this had to be fixed, and turned up the most staggering thing I have seen in years - the RS232 transmission from Easy Script uses the NTSC baud rate timing table, even on PAL machines!

Describing the code for RS232 handling in the C64 as primitive is an act of extreme understatement.

For the record, the offending code is as follows:

```
EF2E LDA  #$40   ;set DSR missing
EF30 BIT         ;skip
EF31 LDA  #$10   ;set CTS missing
EF33 ORA $0297
EF36 STA  $0297 ;set bits in RSSTAT
```

. . .then falls into $EF39, which is the destination of the BEQ at $EF24 for buffer start = buffer end (i.e. empty) – this then shuts down NMIs, which means no further transmission. To maintain service of CTS/DSR line monitoring, it is essential to jump around this code. I have done:

```
EF36 JMP  $FF43

FF43 STA  $0297 ;as original
FF46 RTS        ;avoid RSSTAT change
```

Clearly $FF43 isn't available to tape users! The $AA (TAX) area at $E4B7 to $E4D9 is suitable. Incidentally, if you want to implement 4800 baud transmission, you will need to put the following:

```
E500 $37        ;3600 lo
E501 $00        ;3600 hi
E502 $02        ;4800 lo
E503 $00        ;4800 hi
```

for the PAL table. The original code $E500–$E504 can then be relocated at $FFF3 to $FFF7. As Easy Script uses NTSC, this doesn't help you there, BUT you can fudge the 4800 into the NTSC table ($FEC2–$FED4) in place of a rate you don't use, e.g. 134.5, and call it up appropriately in Easy Script or whatever program you are using.

If anyone would wants to pursue this matter with me, I would welcome correspondence at: 94 Grove Road, Lesmurdie 6076, Western Australia

Peter Morgan

TransBloopers

Low Cost Universal EPROM Programmer

A couple minor problems have surfaced here. . . nothing that would go unnoticed for long though.

In the schematic on page 48, pin 14 of the 8255 IC to the left of the diagram shows two pins numbered 14. The GND pin at the bottom should be pin 7 (same as the GND pin of U2, the 8255 at the right of the diagram).

There are also two pins numbered 14 on the ZIF Socket. The GND pin is correct. The other pin 14 should be 15, pin 15 goes to 16, 17 to 18, and 18 to 19 (i.e. add 1 to each from 14 through 18).

The personality socket for the 2716 on page 47 shows pin 12 going to 9. Correct this so that pin 12 goes to pin 21 (3 down from 24). Pin 9 correctly goes to pin 20 as shown.

Thanks to Ghislain Lamothe or Montreal, QUE. for these corrections.

Frank J. Hermann of Kitchener, Ont. also notes two little typesetting errors in the software. In lines 2760 and 2770, the " should be replaced by " " (ie. null string).

Keyboard Expander 64: Vol. 7, Issue 3

In the program "scroll.obj", the first loop READs 741 data elements to ensure the checksum is correct. The second loop then attempts to write 2745 data elements to a disk file. Change line 1070 to:

```
1070 FOR J = 1 TO 741: READ X
```

Our thanks to Dave Aulbertsberg, MCAS Kuneohe, Hawaii, for pointing out this error, and our apologies for the slip-up.

TeleColumn #2

iNet 2000

"iNet" stands for the "Intelligent Network", a service of Telecom Canada which is an association of 10 of the country's major telecommunications companies. It originally began as a "gateway service" to provide simplified access to public databases around North America. For those accessing several of the participating databases, only one bill would be issued by iNet summarizing all the online charges. iNet then expanded the service adding teleconferencing, electronic mail, user data workspace, editing of retrieved information, and boolean searching capabilities.

Since iNet 2000 belongs to Telecom Canada, access to iNet is done mainly through Datapac, another Telecom Canada project, and the main packet-switching network in Canada. You call your nearest Datapac public dial port (or node) and enter the call address for the iNet service much like entering the call address for any other service accessible from Datapac.

So why pay iNet to transfer you to another database when you can simply go directly there? Remember, Telecom Canada is marketing this service to people rather unlike the typical micro-computer enthusiast (ie. "not us"). It's the difference between business and pleasure. When you're in it to try and make (or save) a buck, you don't care *who* has the information you need, you just need it! You also don't care what it costs to get (which is obvious by the hourly rates of some iNet affiliated databases). Remembering several long call addresses that are often a mixture of numbers and letters doesn't matter much to "us", much like "we" probably wouldn't care about spending a little time handling several online bills if it meant saving money.

However, probably the most downplayed feature of the service is the ability to access it through toll free 1-800 numbers. This isn't surprising though. Most of iNet's target market is probably within a local call of a Datapac public dial port. But! If you're paying long distance charges to get to the nearest dial port, this one feature makes iNet truly attractive.

To subscribe to iNet costs $50.00. You get a really nice manual and some handy reference cards. After that it's $3.00 per month, plus your time while connected to iNet. However, once you pass through the gateway to another service, the iNet meter stops ticking, and you pay the regular charges of the service you access, plus the Datapac surcharge which even local callers pay. When you log off that service, you're back in iNet and the clock resumes. iNet charges range from $15 per hour in prime time (6am to 6pm weekdays) to $11.25 per hour outside these times and weekends. Once you get to know the commands for passing in and out of the gateway, accessing your favourite online service will probably cost you less than a dollar in iNet charges.

Even though iNet is available in the U.S. through Telenet, the toll-free lines only apply to Canada. Ideally, a parallel service is what's needed for telecomputing from remote areas in the U.S. If someone finds one, we'll be most pleased to hear about it.

One other point: iNet, and the databases you can access from iNet, are generally of the text based variety. These are services like CompuServe, Delphi, The Source, or any database that can be controlled using any standard communications program. Systems like Quantum Link and Playnet require the use of their own software which usually handles the sign-on procedure for you. These programs would need to be "aware" of the extra step at the iNet gateway stage. It wouldn't be impossible for Q-Link or Playnet to add this capability to their software, but until they do, it's a situation where your terminal must be compatible with both iNet AND their accessible databases.

In summary. . . if you're into telecomputing for the entertainment value, then iNet is probably not for you. But if you live in Canada, and your long distance bill makes your online bill look trivial by comparison, iNet 2000 is definitely worth looking into.

CompuServe

Sometime between writing this article and printing it we'll be starting construction of what we hope will be named "The Commodore MagNet". This will be the Display Area we've been referring to that will host information supplied by magazines publishing Commodore related material.

The section will be listed under CBMNET with the other Commodore Forums. Selecting "Commodore MagNet" will bring up a menu of magazines, of which Transactor will be one. Selecting "Transactor" will display a menu of activities that will be fundamentally the same for all of the participating publications.

Reading articles and ordering subscriptions are two of these functions. You'll also be able to download programs related to the articles, however, the programs will be stored in a Data Library outside the MagNet area.

Once this section is working, there will be a beta-test period before it goes "live". Hopefully by next issue we'll have all the operational details.

Uploading Time is Free!

When uploading your programs to CompuServe, the clock is turned off! CompuServe figures that if you're generous enough to share your programs with others, they'll reciprocate by suspending the meter while you send in your files.

You might think this would precipitate an awful lot of software in the Data Libraries. Well, you'd be right! Since making upload time free, CompuServe has been literally deluged with programs. Here are a couple tips to remember if you're planning to add to the flood.

CompuServe supports four types of transfer protocols for sending and receiving files from their Data Libraries (as explained last issue). However, two of them are used more often than the others. "Xmodem" protocol has been around a long time and is supported in many terminal programs. In fact, just about any of the online services, including many Bulletin Board Systems, that offer software to download will support Xmodem transfers. The other is "B Protocol". This method was invented by CompuServe to include some of the details that were missing in Xmodem.

CompuServe Filenames and Extensions

CompuServe lists all files in their data libraries using a filename (1 to 6 characters max.) followed by a period and a 3 character "extension". The filename you enter is up to you. But the extensions are used to designate what type of file belongs to the name, and there are a few conventions to be aware of, especially when uploading programs.

If Xmodem protocol is used to upload a program, the extension ".BIN" should be entered after the filename. ".IMG" is the abbreviation for an

"image" file uploaded with B protocol. Following these conventions will make life a lot easier for those who download your donations. However, there are exceptions.

The Exceptions

When uploading programs, please follow this convention:

Uploaded With:	Extension
Xmodem	.BIN
B Protocol	.IMG

Naturally there are other kinds of data besides programs.

.TXT Generally describes text files. If it's text specific to one brand of wordprocessor, the description of the file will usually say so.

.DOC Same as .TXT, only the contents are DOCumentation, usually for another program available nearby.

.MEM A Memo file. Uncommon, but used to indicate a short .TXT file that will probably never need to be "printed on paper"

.HLP A Help file. Usually describes how to perform the various functions on CompuServe

.CNF A Conference transcript. When a guest speaker comes online for a formal conference, a sysop will sometimes record the "CO" and upload the file for others who missed it.

.SEQ Describes anything from text files to data intended for use by another program

.ARC An Archived file. .ARC files are files that have been compressed using a program called "ARC" by Chris Smeets. ARC not only compresses the size of a file, but also allows several files to be combined. ARCing large donations means that downloading them will take much less time, especially when several files are needed (such as a program complemented by modules). Use Xmodem to download them, but you'll need the ARC program in order to decompress. As of this printing, ARC220 is the latest version, and is available in the High Level Utilities section of CBMPRG.

Uploading Text

When uploading files that contain text, it must be sent in true ASCII. Like it or not, CompuServe supports other brands of equipment besides Commodore. But Commodore is the only one that doesn't use true ASCII. So many terminal programs written for Commodore machines include a translation from PETSCII to ASCII. This is often done "on-the-fly" during an upload. So if you're uploading text to CompuServe that is in PETSCII format, turn on the translation switch. If you don't, the upload will appear to be working, but a successful download will be impossible. Files that fall into this category are PaperClip or Easyscript files saved as SEQs.

You also need to send Linefeeds after Carriage Returns during an upload. Likewise, most programs implement this as an "ON/OFF" switchable setting. Although another user will still be able to download this file without the Linefeeds, there is another reason.

You don't have to download text files from CompuServe in order to read them. Once in a Data Library, you can use the READ command to display the contents of any file. You can READ a program, which may be interesting, won't make a lot of sense, and is highly recommended for adding up online charges. Or, you can READ the text files. For example, if you're in DL0 of CBMPRG:

```
READ HOW2CO.HLP
```

. . .will display the help file on how to use the Conferencing section. At 300 baud you could probably read it as fast as it's displayed. But at 1200 you might want to open a capture buffer and read it later. However, depending on your particular software, the absence of Linefeed characters may cause each line of text to print over top of the last. Once again, turn Linefeeds (and true ASCII) on when uploading text files.

So even if you have none of the necessary protocols for downloading, you can still obtain text from the DLs. Depending on the quality of your connection you may experience transmission errors. So if a perfect copy is necessary, you may need to use a downloading procedure.

Downloading

First of all, B Protocol and Xmodem have only very slight differences. In fact, you can download a program using Xmodem that's been uploaded with 'B', or vice versa. But if you do, there are some "post-download" adjustments you'll need to perform. Here are the four possible situations.

Downloading .BIN Files with Xmodem

With Xmodem, programs are transferred in blocks of 128 bytes each. However, most programs won't be evenly divisible by 128 bytes. So the last block is padded to make it fill out to 128. The padding character is a CTRL-z, or CHR$(26). Usually this won't affect the operation of the program. If it was a text file and you load it into your wordprocessor, you'll see the characters at the end and just delete them. But if it's a BASIC program, it means that the variable table will start just a little bit higher in memory. Usually this won't matter. But sometimes it could be a non-relocatable machine code program or something equally as sensitive. The extra bytes at the end might cause the program to over-shoot its intended memory area and write into CIA registers or some other program.

There is a program for dealing with files that are sensitive to the padding in the last block. "XSTRIP.BIN" is in Data Library 8 of the CBMCOM Forum. All it does is ask you for a filename and proceed to strip the CTRL z's off the end of the file. This is done off-line using the file as downloaded on your drive.

Downloading .IMG Files with B Protocol

Variable block size is one reason B Protocol was invented. If B Protocol is used to download a .IMG file (ie. a file uploaded with B Protocol), you should have no problems.

Another reason B was invented was to deal with the different types a files that exist on disk such as PRG and SEQ. This information is contained in a 6 byte header that precedes the remainder of the data. Part of the header is used to indicate that B Protocol is in effect. So when using B to download a .IMG file, the transfer program will detect that the file was uploaded with B, and the first 6 bytes will not be written to your disk drive.

Downloading .IMG with Xmodem

You may not have B Protocol, but you can still download .IMG files with Xmodem. However, those first 6 bytes will be received as valid data and get written to your disk. If it's a program you're downloading, the entire program will be "skewed" by 6 bytes when you LOAD it. This won't do. Therefore, another post-download utility was written called "BINIMG.BIN" and it's also in DL8 of the CBMCOM Forum. All it does is "eat" the first 6 bytes off the file, and write the rest back out.

If you're using CBTerm, this is done automatically. Other authors of terminal programs are also starting to add this feature. The easiest way to check if a BINIMG.BIN adjustment is necessary is to LOAD and RUN the program. If it's a BASIC program and you get a Syntax Error in some strange line number, it needs the 6 byte header removed.

Downloading .BIN with B Protocol

B Protocol "looks" for that 6 byte header. If it's not there, B won't discard those bytes. They will get written to disk and the program will simply proceed with the download much like Xmodem would. You won't need to

worry about the padding characters at the end either. However, the file type will default to SEQ on your Commodore drive.

There are two ways to remedy this. When the program asks you for a filename to use on your disk, follow with ",P". This will force a PRG type file. If you forget and you end up with a program that's locked inside an SEQ file, simply add ",S" to your filename when you LOAD it. This will avoid a ?File Type Mismatch error. Once the program is loaded, you can Scratch the old file, and SAVE a new one.

Choose Your Weapons Carefully

You may have noticed the difference in connect time charges for 300 and 1200 baud. Choosing one or the other for certain tasks can help make your online fun more economical. You can't change "mid-stream", so this decision must be made before you sign-on

1200 baud is more expensive, but if you're signing on with the intention of downloading several programs, 1200 will often save you money in the long run.

However, it's pretty hard to read text flying by at 120 characters per second. It's also pretty tough to type that fast. So for doing things like reading or leaving messages, browsing through DLs, or Conferencing, use 300 baud to save yourself a little money.

CIS Directory

The following list is a summary of all the different sections on CompuServe along with their quick reference words – especially handy for "touring". A "$" indicates an extra charge for the service, and (E) indicates an Executive Information Service option.

Next Issue

Color Mail is one of CompuServe's most advanced offerings. With the Color Mail package you can create animated greeting cards to send to others. You don't need the Color Mail program to receive them – an off-line decoder lets you see your animated mail.

1-800-Floppys	**DSK**
AAMSI Communications	**AAM**
ABC Worldwide Hotel Guide	**ABC**
ACOG	**ACOG**
ADCIS Forum	**ADCIS**
AESNET	**AESNET**
AI EXPERT Forum	**AIE-100**
AI EXPERT Magazine	**AIE**
AMS/OIL Dealer	**AMS**
AOPA Forum ($)	**AOPA**
AP Datastream	**SPD-1005**
AP Videotex, Business	**APV**
AP Videotex, Entertainment	**APV**
AP Videotex, Politics	**APV**
AP Videotex, Weather	**APV**
AP Videotex, World News	**APV**
ASI Monitor	**ASI-10**
ASI Service Difficulty Reports	**ASI-12**
Academic Amer. Ency ($)	**ENCYCLOPEDIA**
Access (Public File Area)	**ACCESS**
Access Phone Numbers	**PHONE**
Adventures in Travel	**AIT**
Agri-Commodities	**ACI**
Air France	**AF**
Aircraft	**AVIATION**
Alaska Teleshopper	**AK**
American Airlines	**AA**
American College of OB/GYN	**ACO**
American Express	**AXM**
American Express(R) ADVANCE	**AXP**
American Tire Buyers	**ATB**
Ameropa Travel	**AT**
Amiga Forum	**AMIGAFORUM**
Antic Magazine	**ANTIC**
Apparel Concepts for Men	**APC**
Apple User Groups Forum	**APPUG**
Ashton-Tate Forum	**ASHFORUM**
Ashton-Tate Support Library	**ASHTON**
Ask Mr. Fed Forum ($)	**MMS-20**
Associated Press	**APN**
Astrology	**GAM-45**
Astronomy Forum	**ASTROFORUM**
Atari 16 Bit Forum	**ATARI16**
Atari 8 Bit Forum	**ATARI8**
Atari Developers Forum	**ATARIDEV**
Athlete's Outfitters	**ATH**
Auto Racing Forum	**RACING**
Autodesk Software Forum	**ADESK**
Aviation Forum (AVSIG)	**AVSIG**
Aviation Menu	**AVIATION**
Aviation Safety Institute	**ASI**
Bacchus Wine Forum	**WINEFORUM**
Baffle Word Game	**BAFFLE**
Banking Services	**BANKING**
Banshi	**BANSHI**
Bantam Books	**BB**
Beneficial National Bank	**BNB**
Biorhythms	**BIORHYTHMS**
Birkenstock Footwear	**BF**
BlackDragon	**BLACKDRAGON**
Bloomingdale's By Mail	**BL**
Bonds Listing ($)	**MMM-41**
Borland International	**BORLAND**
Braille	**BRAILLE**
Broadcast Professionals Forum	**BPFORUM**
Buick Magazine	**BU**
Business Incorporating Guide	**INC**
CB Interest Group	**CBIG**
CB Pictures	**CBPIX**
CB Society	**CUPCAKE**
CBS/Fox Video	**CF**
CP/M (CPM) Users Group	**CPMSIG**
CW Communications	**CW**
Calculate Net Worth	**HOM-16**
Careers	**WS**
Carolina Health & Fitness	**HF**
Casino	**CASINO**
Castle Telengard Game	**CASTLE**
CastleQuest	**CQUEST**
Casual Tee's	**CA**
Changing Your Password	**PASSWORD**
Changing Your Terminal Type	**TERMINAL**
Checkbook balancer	**CHECKBOOK**
Chevy Showroom	**CHV**
Christian Book Store	**DII**
Citibank	**CI**
Citizens Band Simulator	**CB-10**
Classic Quotes	**TMC-7**
Coffee Emporium	**COF**
College Press Service	**CPS**
Colonial National Bank USA	**CN**
Color Graphics	**CIS-91**
Color Mail Database	**COLORMAIL**
Color Mail Exchange Forum	**HALLMARK**
Comic Book Forum	**COMIC**
Command Decision	**COMDEC**
Commodities	**COMMODITIES**
Commodity Pricing ($)	**CPRICE**
Commodity Symbol Lookup	**CSYMBOL**
Commodore Arts and Games Forum	**CBMART**
Commodore Communications Forum	**CBMCOM**
Commodore Programming Forum	**CBMPRG**
Commodore Service Forum	**CBM-2000**
Commodore Users Network	**CBMNET**
Comp-U-Store	**CUS**
Compu-Game	**CPG**
CompuServe Billing Information	**BILLING**
CompuServe Command Summary	**COMMAND**
CompuServe Logon Instructions	**LOGON**
CompuServe Node Abbreviations	**NODES**
CompuServe Rates	**RATES**
CompuServe Tour	**TOUR**
CompuServe's Product Ordering	**ORDER**
CompuServe's Subject Index	**TOPIC**
CompuServe's software exchange	**SOFTEX**
Computer Art Forum	**ARTSIG**
Computer Club Forum	**CLUB**
Computer Consultant's Forum	**CONSULT**
Computer Express	**CE**
Computer Language Magazine	**CLM**
Computer Sports World	**CSW**
Computers/Actrix/Eagle/Timex	**CLUB**
Computing Tutorials	**PCS-121**
Conroy-Lapointe	**CL**
Consumer Electronics Forum	**CEFORUM**
Cooks Online Forum	**COOK**
Cosmic Concepts	**CC**
Current Day Quotes ($)	**QQUOTE**
DEC PC Forum	**DECPC**
DISCLOSURE II ($E)	**DISCLOSURE**
DR. JOB	**DRJ**
DataPac Logon Instructions	**LOG-41**
Department of State	**STATE**
Detailed Issue Examination ($)	**MMM-39**
Digital Equipment Corporation	**DECUNET**
Digital Research Forum	**DRFORUM**
Digital Research Inc.	**DRI**
Directory Of Public Officials	**OFFICIALS**
Disabilities Forum	**DISABILITIES**
Discount Computers	**DSC**
Discover Orlando	**ORLANDO**
Dividends and Splits ($)	**DIVIDENDS**
Donoghue Organization	**DON**
Dow Jones & Co.	**DJ**
Download Pricing Data	**MMM-67**
Dr Dobb's Journal	**DDJ**
Dr. Dobb's Journal Forum	**DDJFORUM**
EF Hutton	**EF**
EMI Aviation Services ($)	**EMI**
EMI PRO-PLAN Registration	**PROPLAN**
EMI RNAV/LORAN Flight Plan ($)	**AERONAV**
EMI Radar Map ($)	**AERORAD**
EMI User Route Flight Plan ($)	**AEROROUTE**
EMI VOR/Airway Flight Plan ($)	**AEROVOR**
EMI Weather Briefing ($)	**AEROBRIEF**
EPIE Database	**EPI**
EPIE Forum	**EPIEFORUM**
EasyPlex	**MCIMAIL**
EasyPlex Electronic Mail	**EASYPLEX**
Ebsco Magazine Entree	**ME**
Ecopress Periodicals	**ECO**
EdVENT II Seminar Directory	**EDVENT**
Educational Research Forum	**EDRESEARCH**
Educational Travel Connection	**EDTRAVEL**
Educators Forum	**EDFORUM**
Electronic Bounce Back	**EBB**
Electronic Gadget Store	**EGS**
Electronic Gourmet ($)	**GOURMET**
Electronic's Mart	**ELM**
Epson Forum	**EPSON**
Equitable Life	**EL**
Executive Engravers	**EX**
Executive News Service ($E)	**ENS**
Executive Option	**EXECUTIVE**
Expert Investor/MQuote II ($)	**MMM-19**
Express Music CDs	**EMC**
FBI Ten Most Wanted List	**TEN**
Family Computing Electronic Ed	**FAM**
Family Computing Forum	**FAMFORUM**
Feedback to CompuServe	**FEEDBACK**
Fifth Avenue Shopper	**FTH**
Financial Documentation	**IQH**
Financial File Transfer	**FILTRN**
Financial Forecasts	**EARNINGS**

Financial Surcharge List	**MMM-23**
Florida Forum	**FLORIDA**
Florida Fruit Shippers	**FFS**
Flying	**AVIATION**
Flying Buffalo	**BUFFALO**
Football	**FOOTBALL**
Foreign Language Forum	**FLEFO**
Fort Worth Computer Chronicles	**FWCC**
Forth Forum/Creative Solutions	**FORTH**
Forums	**FORUMS**
Futures Focus Stock Index Plus	**FFP**
Gamers Forum	**GAMERS**
GlobaLink	**GLO**
Golf	**GOLF**
Good Earth Forum	**GOODEARTH**
Government Publications	**GPO**
HAMNET	**HAMNET**
Hallmark Color Mail	**COLORMAIL**
HamNet Online	**HAM**
Handicapped Users' Database	**HUD**
Hangman	**HANGMAN**
Hardware Forums	**PCS-20**
Hawaiian Isle	**HI**
Health Forum	**HCM-660**
HealthCom	**HCM**
HealthNet	**HNT**
Heath Users Group	**HEATHUSERS**
Hewlett Packard Forum	**HP**
Historical Pricing	**PRICES**
Hobbit Hole / Wyandotte Wines	**HH**
Hollywood Hotline ($)	**HOLLYWOOD**
Hollywood Hotline Art	**HHA**
Home Banking	**BANKING**
Human Sexuality	**HUMAN**
Huntington National Bank	**HNB**
I/B/E/S ($E)	**IBES**
IBES Earnings Estimate Reports	**IBES**
IBM Communications Forum	**IBMCOM**
IBM Hardware Forum	**IBMHW**
IBM Junior Forum	**IBMJR**
IBM New Users Forum	**IBMNEW**
IBM Software Forum	**IBMSW**
IBM Users Network	**IBMNET**
IIAA	**INS**
IQuest	**IQUEST**
Incue Online	**INCUE**
Independent Insurance	**INS**
Information USA	**IUS**
Intelligence Test	**TMC-32**
Internal Revenue Services	**IRS**
International Fur Wholesalers	**RF**
Investors Forum	**INVFORUM**
Island of Kesmai	**ISLAND**
Journalism Forum	**JFORUM**
Kaypro Users Forum	**KAYPRO**
LDOS/TRSDOS6 Users Group	**LDOS**
LOGO Forum	**LOGOFORUM**
Legal Forum	**LAWSIG**
Lincoln Manor Baskets	**LM**
Literary Forum	**LITFORUM**
Living Videotext Forum	**LVTFORUM**
Lobster Market	**SEA**
Logical Systems Inc Forum	**PCS-49**
MAUG(tm)	**MAUG**
MAUG(tm) Apple II & III Forum	**APPLE**
MAUG(tm) Apples Online	**AOL**
MAUG(tm) MacDeveloper's Forum	**MACDEV**
MAUG(tm) Macintosh Users Group	**MACUS**
MEDSIG	**MEDSIG**
MMS/Daily Comment ($)	**DC**
MMS/Fedwatch Newsletter ($)	**FW**
MMS/Market Briefings ($)	**MAR**
MUSUS Forum	**MUSUS**
Magic Castle Video	**MV**
Market Highlights ($)	**MMM-46**
MaryMac Industries, Inc.	**MM**
Max Ule Discount Brokerage	**MU**
Max Ule's Mergersource	**TKR**
Max Ule's Tickerscreen	**TKR**
MegaWars I	**MEGAI**
MegaWars I Pictures	**MW1PIC**
MegaWars III	**MEGAIII**
Mercury House	**MER**
MicroPro Forum	**MICROPRO**
MicroQuote ($)	**MQUOTE**
Microsearch Reference Library	**MSH**
Microsoft Forum	**MSOFT**
Military Veterans Services	**VET**
Milkins Jewelers	**MJ**
Minutiae Challenge	**MINUTIAE**
Misco Computer Supplies	**MO**
Model Aviation Forum	**MODELNET**
Money Market Services	**MMS**
Monthly Charges	**MONTH**
Morrow's Nut House	**NUT**
Mortgage Calculator	**HOM-17**
Movie Reviewettes	**MOVIES**
Multi Issue Price History ($)	**QSHEET**
Multi-Player Games Forum	**MPGAMES**
Music Alley Online	**MAO**
Music Forum	**MUSICFORUM**
Music Video	**MUS**
NWS Aviation Weather ($)	**AWX**
NWS Weather	**WEA**
Naked Eye Astronomy	**NAKEDEYE**
National Bulletin Board	**BULLETIN**
National Issues Forum	**ISSUESFORUM**
National Tourism Citilog	**CITIES**
Nationwide Catalog Shopper	**NCS**
Neighborhood Demographics ($)	**NEIGHBOR**
Neiman-Marcus	**NM**
New Adventure	**NEWADVENT**
New Car Showroom	**NEWCAR**
News-A-Tron	**NAT**
Newsnet	**NN**
Noload Mutual Funds Directory	**NOLOAD**
OB-GYN	**OBG**
OMNI FORUM	**OMNIFORUM**
OMNI Online	**OMNI**
OP-NET Forum	**SFP-4**
OS9 Forum	**OS9**
Official Airline Guide EE	**OAG**
Official Airline Guides	**OA**
Ohio Scientific Forum	**OSIFORUM**
Online Computer Connection	**COMPUSERVE**
Online Today	**ONLINE**
Options Profile	**OPRICE**
Orch-90 Music Forum	**ORCH-90**
Original Adventure	**ORADVENT**
Outdoor Forum	**OUTDOORFORUM**
PDP-11 Forum	**PDP11**
PLASTISERV Plastics Inform	**SFP-13**
PR Link	**PRLINK**
PR and Marketing Forum	**PRSIG**
PSFS Direct Line Banking	**PSFS**
Pan Am Travel Guide	**PANAM**
PaperChase-MEDLINE	**PAPERCHASE**
Pascal Forum	**PCS-55**
Personal Computing	**COMPUTERS**
Personal File Area	**FILES**
Personal Menu	**MENU**
Personality Profile	**TMC-17**
Peterson's College Guide	**PETERSON'S**
Pictures Support Forum	**PICS**
Plastics Associations	**DIR**
Plastics Buyer's Guides	**BUY**
Plastics Directories	**DIR**
Plastics Government Agencies	**DIR**
Plastics Legislation	**SPI**
Plastics Materials	**PRO**
Plastics Product Guide	**PRO**
Plastics Regulatory Update	**REG**
Plastics Statistics	**SPI**
Plastics Suppliers	**BUY**
Portfolio Valuation ($)	**PORT**
PowerSoft's XTRA-80	**PCS-56**
Pricing Statistics ($)	**STATS**
Programmers Forum	**PROGSIG**
Public Access	**ACCESS**
Question & Answer	**QUESTIONS**
Quick Quote ($)	**QQUOTE**
Quick Reference List	**QUICK**
Quick Way	**QWKWAY**
RCA Direct Marketing	**RC**
Rapaport Diamond Broker	**RDC**
Rare Disease Database	**RDB**
Record World	**RW**
Religion Forum	**HOM-33**
Return Analysis ($E)	**RETURN**
Rin Robyn Pool & Patio	**RR**
RockNet	**ROCK**
Rocky Mountain Connections	**ROCKIES**
SBENET	**SBENET**
SHOWBIZ Quiz	**SHOWBIZ**
Safetynet Forum	**SAFETY**
Sailing Forum	**SAILING**
Savings Scan	**SAV**
Science Fiction/Fantasy	**SCI**
Science Trivia Quiz	**SCITRIVIA**
Science/Math Education Forum	**SCIENCE**
Scott Adams' Games	**ADAMS**
SeaWAR	**SEAWAR**
Sears, Roebuck & Co.	**SR**
Securities Screening ($E)	**SCREEN**
Shareholders Freebies	**FRE**
Shawmut Bank of Boston	**SHW**
Shop-at-home	**SHO**
Simon David	**SIM**
Single Issue Price History ($)	**PRICES**
Soap Opera Summaries	**SOAPS**
Social Security Administration	**SSA**
Society of Plastics Industry	**SPI**
Software Discounters of Amer	**SDA**
Software Publishing Forum	**SPCFORUM**
Software Publishing Online	**SPC**
Southeast Bank	**SEB**
Space Forum	**SPACEFORUM**
SpaceWAR	**SPACEWAR**
Sports Forum	**HOM-110**
Standard & Poor's ($)	**S&P**
Standard and Poor's ($)	**S&P**
State Capitol Quiz	**TMC-44**
Stevens Business Reports	**SBR**
Students' Forum	**STUFO**
Subscribers Directory	**USERS**
Sun Life Group	**SLG**
Sun N Sand Vacations	**SNS**
Sunland Camera	**SUN**
SuperSite ($E)	**DEMOGRAPHICS**
TELPLT ($)	**MMM-9**
THE ELECTRONIC GAMER(tm)	**EGAMER**
TRS-80 Model 100 Forum	**M100SIG**
TRS-80 Professional Forum	**TRS80PRO**
TYMNET logon instructions	**LOG-11**
Tandy Business Users Group	**TCBUG**
Tandy Color Computer Forum	**COCO**
Tandy Newsletter	**TRS**
TeleData•Guide	**TDG**
Telecommunications Forum	**TELECOMM**
Telenet Logon Instructions	**LOG-20**
Texas Instruments Forum	**TIFORUM**
Texas Instruments News	**TINEWS**
Text Editors	**PCS-121**
The Business Wire	**TBW**
The College Board	**TCB**
The Electronic MALL	**MALL**
The Game Getters, Inc.	**GG**
The Grower's Store	**SDG**
The Heath Co.	**HTH**
The McGraw-Hill Book Company	**MH**
The Multiple Choice	**TMC**
The National Satirist	**KCS**
The Tandy Users Network	**TANDYNET**
The Whiz Quiz	**WHIZ**
The World of Lotus	**LOTUS**
Ticker Retrieval ($E)	**TICKER**
Ticker and Cusip Lookup ($E)	**TICKER**
Tiffany & Co.	**TIF**
Topgar Tobaccos	**TG**
Touch-Type Tutor	**TMC**
Tour the West	**WEST**
Travel SIG	**TRAVSIG**
TravelVision	**TRV**
Traveler's Challenge	**ETC-81**
Travelshopper	**TWA**
Tropical Fish Forum	**FISHNET**
US Entrepreneurs' Network	**USEN**
USA TODAY	**US**
Unified Management	**UMC**
United American Bank	**HOM-152**
VAX Forum	**VAXSIG**
VIDPLT ($)	**MMM-47**
VIDTEX Information	**VIDTEX**
VIDTEX Weather Maps	**MAPS**
Vacuum Advance	**VCS**
Value Line Financials	**MMM-10**
Value Line Information	**VLINFO**
Value Line Projections ($)	**MMM-7**
Vermont Tourism	**VERMONT**
Videolog Electronics	**VL**
Visa Advisors	**VISA**
VitaMenagerie	**VM**
WITSIG	**WITSIG**
Waldenbooks	**WB**
Walter Knoll Florist	**WK**
Wayside Systems	**WS**
West Coast Travel	**WESTCOAST**
What's New	**NEW**
What's New in Travel	**WNT**
Whole Earth Software Forum	**WHOLEEARTH**
Woodstock Leather	**BAG**
Word Scramble	**SCRAMBLE**
Working-From-Home Forum	**WORK**
World of Computers	**WOC**
World-Wide Investment Systems	**REAL ESTATE**
Worldwide Property Guide	**WWX**
Writers and Editors Forum	**WESIG**
Xerox Direct Marketing	**XDM**
You Guessed It!	**YGI**

An Introduction to Machine Language Programming on the Amiga

Rick Morris, Burnaby, British Columbia

After purchasing my Amiga and ogling some demo programs, I decided it was time to start writing software myself. I had bought the developer's kit, and included in it was plenty of documentation on how to use the various system routines and libraries. Unfortunately, the examples were all in 'C', a language I did not know. Learning C and the Amiga environment proved to difficult, so I switched languages to Assembler, something I have had some experience with.

The program that follows is a demonstration of how to access system routines with Assembler. In itself it doesn't do much, and is not a marvelous example of programming, but it provides basic text input–output and forms the basis of a Command Line Interface I am writing, taking advantage of the Function keys and Menus. The program sections are numbered in the right–hand margin, and I will refer to the sections by number. Further, I will not go into great detail about the specifics of each system routine, instead I refer you to the AmigaDOS Developer's Manual.

Section 1 is System Equates, and most of them are found in the assembler INCLUDE files. I find it best to put them right in the program, since a considerable amount of time is taken up when you include any files. If you must use the include files, I suggest you at least strip the comments out of them. Alternately, you can assemble the include files with the –e option, and include the resultant file as you would any other include file. This method saves the time the assembler normally takes to expand the regular include files.

Section 2 tells the assembler that these 'xref'ed labels will be resolved at link time, and are found in the "amiga.lib" file.

Section 3 is a Macro that saves me from typing the dreaded __LVO each time I want to use one of the system calls.

Section 4 is the start of the program. I call the beginning of the program __main from habit, and if you link in "Astartup.obj" as well as "amiga.lib" when you ALink the object code, then a routine will be included that allows your program to be run from either CLI or WorkBench, and the label __main must be at the start of your code.

Here is where we use our first call to the system routines, the call OpenLibrary(a6). A short discussion of the way the Intuition is set up is in order.

There are 2 basic types of system calls, Exec routines and Libraries. All of the routines are accessed through jump tables. You load the address of the beginning of the jump table into address register a6, information required by the routine in the other registers, and then you JSR __LVORoutine(a6). The routine performs the asked–for function and returns a value, which may be an error code that you must test for. Our __LVOOpenLibrary (or any other system call) is turned into an offset at ALink time, when you link with amiga.lib. For example, the Assembler syntax JSR __LVOOpenLibrary(a6) becomes, after linking, JSR $FFFE68(a6).

The starting address of the Exec jump table is contained in Memory Location $000004, AbsExecBase, the only unchanging memory location in the machine. This jump table contains system routines, such as OpenLibrary, AllocMem, and all of the routines found in the Exec/doc section of the Rom Kernal Manual. Each time you use one of these routines you must have the pointer you saved from Location $000004 in address register 6.

The next level of routines dealt with in the example is Library routines. We call OpenLibrary to open the "dos.library" in order to use dos routines such as Read and Write. They are used in the same way that system routines are used. Our call to OpenLibrary returned the address of the dos library jump table, which we then store for later use. Note that zero may be returned indicating an error, and we must test for this. Other libraries, which may be disk–based and loaded into memory only when opened, include the Translator, the Font library, and the Math library. These libraries are used in the same way as the Dos library: you call OpenLibrary with a pointer to the name of the library, then use the pointer to the jump table returned.

Section 5 uses one of the DOS routines to open a window, in this case RAW:. This window is actually a file that does not pre–process keystrokes, so we can get all keys, including cursor

keys and function keys. You use the same process to open disk files, and if the file type is MODE_NEWFILE, a new disk file is opened for you to write to.

The parameter returned from Open is the 'handle' of the file, and is one of the parameters we in turn pass to Read and Write.

The 'tst.l d0' instruction used here may appear redundant, and in fact is if we re-arrange the instructions so that the 'move.l d0,handle' comes before the 'tst.l d0' instruction, but illustrates a point. The flags are not necessarily set upon return from a system call, and you must explicitly test results yourself.

Sections 6 and 7 are housekeeping for pointers and printing a 'Hello' to the window we have just opened.

Section 8 is the logic of our program. It gets a keystroke, and decides upon appropriate action. An 'escape' key ends the program, a 'return' executes the CLI command just typed in, and any other key is echoed to the window.

Section 9 gets keys from our window. The WaitForChar routine will wait for 500 microseconds, or until a key is pressed, and return 0 for no key, -1 for a key pressed. When -1 is returned, we Read the key into our buffer.

Once again, we seem to have a redundant routine. We could just as easily have used Read, and let it wait for us until a key was pressed. The reason we use WaitForChar is that this the Amiga is a multitasking machine, and if no input is coming our way, the WaitForChar allows the system to put our task to sleep and service other routines, then wakes us up when a key is pressed. It is always a good idea to let go of system resources when you do not need them.

Section 10 prints the last key pressed. Notice that we put d0 on the stack and get it back before exiting the routine. Data registers 0 and 1, and Address registers 0 and 1 are scratch to all system routines, and all other registers will be preserved. Or at least Commodore has stated that they will be when Kickstart 1.2 arrives.

Section 11 executes any command we type in before a 'return'. This is almost ridiculously simple: we put the command string address in d1 (not a1, strange but true), zero out d2 to say there is only one command, put our File Handle in d3, and call Execute - Dos does the rest. Any command that can be typed in at the CLI prompt can be used, and the results will be sent to our window.

Section 12 is called by Section 7, and prints our 'Hello' to the newly opened window. Note that since we are using a RAW: window, we must include line feeds (#10) in our message string.

Section 13 closes our window, then the dos library. You should always close any resource you open. This will give back the memory the system reserved for you when you opened it, and in the case of disk-based libraries, will allow the RAM used by the library to be reclaimed.

Section 14 defines some strings used elsewhere in the program.

Section 15 also defines storage, but uninitialized. The 'section bss' directive tells the assembler we need this amount of storage, but it is not included in the assembly. At load time, the system automatically sets aside this amount of free storage for us, rather than loading in a bunch of blank bytes.

I hope this example is helpful in starting others on Amiga assembler programming. I have found that C Structures make much more sense as ML data tables, and the system routines available are quite easy to use and very powerful. And, of course, the 68000 processor has a wonderful variety of instructions and addressing modes available.

The assembly starts here.

```
* some system equates
AbsExecBase     equ $4          *AbsExecBase                    (1)
timeout         equ $500        *how long we wait for input
rtn             equ $d          *ascii carriage return
esc             equ $1b         *ascii escape key
MODE_NEWFILE    equ 1006        *file mode to create a new file


                xref _LVOOpenLibrary                            (2)
                xref _LVOCloseLibrary
                xref _LVOOpen
                xref _LVORead
                xref _LVOWrite
                xref _LVOWaitForChar
                xref _LVOExecute
                xref _LVOClose
                xref _LVOInput
                xref _LVOOutput

call    MACRO                   *a macro to save typing         (3)
        jsr     _LVO\1
        ENDM

*open DOS                                                       (4)
__main  move.l  AbsExecBase,a6  *pointer to exec
        move.l  a6,execbase     *  and store it
        lea     dosname(pc),a1  *pointer to dos.library
        moveq   #0,d0           *version no.
        call    OpenLibrary(a6) *open
        move.l  d0,dosbase      *save dosbase
        beq     close           *library didn't open

*Open the console device                                        (5)
        lea     conname(pc),a1  *name in d1?
        move.l  a1,d1           *move address to d1
```

```
        move.l  #MODE__NEWFILE,d2  *file type is newfile
        move.l  dosbase,a6         *get dosbase
        call    Open(a6)           *open console
        tst.l   d0                 *check error
        beq     close2
        move.l  d0,handle

*initialize the variables                                    (6)
        lea     buffer,a1          *initialize the buffer
        move.l  a1,bufptr          *   pointer

*print the welcome message                                   (7)
        jsr     mess               *print a welcome message

* the start of the routines                                  (8)
forever bsr     key                *get a key
        cmp.b   #esc,d0            *is it the escape key?
        beq     close1             *exit
        bsr     print              *print the character
        cmp.b   #rtn,d0            *is it a c/r?
        beq     docmd              *execute the command
        addi.l  #1,bufptr          *increment the buffer pointer
        bra     forever            *continue the loop

*get a key without testing it                                (9)
key     move.l  handle,d1          *handle for the console
        move.l  #timeout,d2        *timeout value
        move.l  dosbase,a6         *get the library pointer
        call    WaitForChar(a6)    *any keys in the input stream?
        tst.l   d0                 *no keys is 0
        beq     key                *wait for a key press

        move.l  handle,d1          *handle for the console
        move.l  bufptr,d2          *load buffer pointer
        move.l  #1,d3              *read one character
        move.l  dosbase,a6         *
        call    Read(a6)           *do the read function
        move.l  bufptr,a1          *restore pointer
        move.b  (a1),d0            *return char in d0
        rts

*print the last key pressed                                   (10)
print   move.l  d0,-(sp)           *remember d0 is scratch!
        move.l  handle,d1          *console handle
        move.l  bufptr,d2          *register usage
        move.l  #1,d3              *one character to write
        move.l  dosbase,a6
        call    Write(a6)          *write one character
        move.l  (sp)+,d0           *get d0 back
        rts

*execute a command                                           (11)

docmd   move.l  bufptr,a1          *get the buffer pointer
        move.b  #10,(a1)           *a line feed character
        jsr     print              *  print it
        move.l  bufptr,a1          *get the buffer pointer
        move.l  a1,d1              *and put into d1
        moveq.l #0,d2              *inhandle is zero
        move.b  d2,(a1)            *zero over c/r
        lea     buffer,a1          *get the buffer address
        move.l  a1,d1              * put it in d1
        move.l  handle,d3          *output handle
        move.l  dosbase,a6         *get the library
        call    Execute(a6)        *Execute the cmd
        lea     buffer,a1          *time to reset the buffer
        move.l  a1,bufptr          * point to start of buffer
        bra     forever

* the welcome message                                         (12)
mess    lea     message(pc),a1     *get the message pointer
        move.l  a1,d2              *put it in d2
        move.l  handle,d1          *our console handle
        move.l  #77,d3             *no. of characters
        move.l  dosbase,a6         *get dos base
        call    Write(a6)          *call system write routine
        rts

*exit program                                                (13)
close1  move.l  handle,d1          * our console is open,
        move.l  dosbase,a6         *  so we must close it
        call    Close(a6)          *close the file
close2  move.l  dosbase,a1         * dos.library is open,
        move.l  execbase,a6        *  so we close it
        call    CloseLibrary(a6)   *close dos.library
close   rts                        *return to cli

*library defines                                             (14)
dosname dc.b    'dos.library',0
        cnop    0,2
conname dc.b    'raw:0/0/640/200/ASM Demo',0
        cnop    0,2
message dc.b    'ASM Demo by Rick Morris',13,10
        dc.b    'Copyright 1986',13,10
        dc.b    'Permission to copy but not to sell',13,10,0
        cnop    0,2

    section     vars,bss                                     (15)

buffer  ds.l    20                 *buffer, 80 bytes
bufptr  ds.l    1                  *current buffer pointer
handle  ds.l    1                  *console file read handle
dosbase ds.l    1                  *dos library pointer
execbase ds.l   1                  *AbsExecBase pointer

    end
```

Amiga Programming Concepts

by Chris Zamara and Nick Sullivan

The Amiga is a complex machine,
but programming it can be easier than you might expect

Part 1: Fundamentals

In this, the first of a series of articles on programming the Amiga, we are going to try to overcome some of the common obstacles that discourage many would–be Amiga programmers. Mostly we'll talk about programming in the C language, with lesser coverage of 68000 Assembly language. We will not attempt to teach you how to program in C or assembler – our feeling is that the best place to learn a language is a good textbook – but merely how to program the Amiga using one of these languages as a tool. Even if C or assembler are not your primary interests, we recommend that you become familiar enough with them that you can follow source code and examples: most documentation and explanations of the Amiga Kernel use examples written in C; knowing a bit of assembler can come in handy if you want to optimize compiled code. Besides a good text on C (The definitive one is "The C Programming language" by Brian Kernighan and Dennis Ritchie), you'll need the Amiga documentation: the Rom Kernel, Dos, and Intuition manuals.

If you're worried that programming the Amiga is impossibly complicated and you're intimidated by the bulk of the manuals you need to wade through, here's a little secret you might be interested in: programming the Amiga is *easy*. However hard it may seem at first, the fact is that the operating system does so much for you that performing major operations is often just like filling in a form – simply assign the values you want to a system structure, then call a built–in function to interpret the values and do your bidding. In this article, we'll cover three important areas that can be most confusing to the new Amiga programmer: structures, include files, and libraries.

Structures

C programmers are familiar with the concept of a structure, which may be thought of as a fancy array that can hold several variables of different types. This section contains ideas that might be redundant to those who already know C, but they are stressed here because they are especially important when programming the Amiga. The Amiga's operating system uses structures heavily as a means of passing packages of data between functions without having to specify each component separately as a variable.

For example, consider the function OpenWindow( ) in the Intuition library. To open a window, you just pass the OpenWindow function a pointer to (that is, the address of) a 'NewWindow' structure that contains the necessary information about the window you want to open. The OpenWindow function knows the definition (called a 'template') for the NewWindow structure, and your program knows too, because it obtained the definition from one of the standard 'include' files (more about those in the next section). The OpenWindow function returns a pointer to an instance of a "Window" structure, which your program can subsequently use to find out things about the window that was just opened (its current width, etc), and which you may also use as an argument to other Intuition functions. Structures are also used throughout the system as a means of communication between tasks, though this is not something that you need to worry about for most applications.

It is important to keep in mind the difference between a structure template and an instance of that structure. For example, we could define a "foo" structure like this:

```
struct foo {
    char name[20];
    unsigned int RefNumber;
    struct foo *NextGuy; /* pointer to another structure */
};
```

Now "struct foo" is a variable type that we can use in declarations. So, if we want to declare two structures of type "foo", we could use the declaration:

```
struct foo Customer, Client;
```

Now "Client" and "Customer" are structures of type "foo". Foo is a structure template, while Client and Customer are instances of the foo structure. Individual members of Client and Customer can now be referenced using the 'dot' operator, like this:

```
n = Client.RefNumber;
```

The templates for system structures are generally defined in the include files that your program uses, and the structures them-

selves are declared in your program. So, if you wanted to make your own NewWindow structure and call it "MyNewWindow", you would use the declaration:

```
struct NewWindow MyNewWindow;
```

Another important distinction to be made is the difference between a pointer to a structure and the structure itself. When using functions that pass information via structures, you usually deal with pointers. When you are dealing with both structures and pointers to structures, it is easy to get mixed up. Don't, because you access the elements of the structure differently in each case. For example, we declared a NewWindow structure above called MyNewWindow; now let's declare a pointer to a "Window" structure and call it "MyWindowPtr":

```
struct Window *MyWindowPtr;
```

We access a member of MyNewWindow like this:

```
MyNewWindow.LeftEdge = 100;
```

. . .and a member of MyWindowPtr like this:

```
x = MyWindowPtr->Width;
```

The arrow operator (->) is used when accessing a structure member from a pointer, and dot (.) is used when using the structure itself. As shown in the "foo" structure example, a structure can contain pointers to other structures; this capability is often used to create a linked list of structures of the same type.

In assembler, structures are simulated by defining the name of each structure element as an offset into the structure. As with C structure templates, these definitions take place in include files. (The manner in which these definitions are done is quite interesting, as they closely simulate C syntax through the use of macros; it's worth taking a look through some include files to see how it's done.) The number of bytes taken by each structure is also defined, so that you can allocate space in your program for any structure you need. This space in your program forms the instance of the structure, and the label you assign to it is the structure name. You can access a member of the structure by using the member name (as defined in the include file) as an offset to the structure name. For example, to "declare" a NewWindow structure called MyWindow, you just make enough space for it (probably at the end of your program) like this:

```
MyWindow    DS.B nw_SIZE
```

Now you can put the address of the structure in a register, like this:

```
LEA MyWindow,a2
```

. . .and store a value into a structure member like this:

```
MOVE #100,nw_Width(a2)
```

That's enough about structures for now – let's get on to the next topic!

Include Files

One thing that many people find hard to cope with when first attempting to write a program on the Amiga is the multitude of INCLUDES that always seems necessary at the top of every source file. In other systems, you can write a lot of programs without including anything other than "stdio.h". On the Amiga, it's a different story, as you'll see if you take a look at the source code of any Amiga–specific program, or through the maze of files in the INCLUDE directory on a C development disk.

Which include files you need to use in your program will largely depend on which libraries and "devices" you are using. For example, if you are using the Intuition library (more about libraries in the next section), you will need the include file "intuition/intuition.h". If you're not sure what you need to include, just use the above guideline and attempt to compile and link. If you get errors indicating that certain structures, constants or macros are not defined, use the SEARCH command to locate its symbol name in the include directory, then INCLUDE the file in which it is found. The filenames of the include files always end in ".i" for assembler, or".h" (for header) for the C programs. Either way, they contain basically the same information. You'll find the following kinds of item:

Defines (constants): These set up system–specific quantities that represent modes, error conditions, flags and so on. When you open a window through Intuition, for example, one of the things you can specify is the set of system gadgets – the size gadget, the drag bar etc. – you want to use. You can do this with a line like:

```
newwindow.flags = WINDOWDRAG | WINDOWCLOSE;
```

which is equivalent to:

```
newwindow.flags = 0x0002 | 0x008;
```

or even:

```
newwindow.flags = 0x000a;
```

The numeric values of WINDOWDRAG and WINDOWCLOSE are defined, along with a host of other things, by #define statements in the include file "include/intuition/intuition.h". You could just use the numbers if you wished to, of course, but that would be foolish – the names of the flags are much easier to remember, and make your program independent of operating system revisions.

Macros: These provide a shorthand method of incorporating often–used pieces of code into your programs. A classic example is the MAX macro:

```
#define MAX(x,y) ( (x) > (y) ? (x) : (y) )
```

Some include files contain macro definitions to facilitate Amiga-specific operations, such as setting bits in system variables. An example from the file "include/graphics/gfxmacros.h":

```
#define SetWrMsk(w,m) {(w)->Mask = m;}
```

Simple macros such as these could be easily coded by hand (e.g. rp->Mask=x; instead of SetWrMsk(rp,x);), but using the provided macros ensures that your program will be compatible with future operating system revisions. The Amiga manuals recommend that you always use the standard macros instead of coding such assignments yourself.

Structures: Structure definitions are one of the main purposes of the include files. The operating system uses structures extensively to maintain all kinds of necessary information about the current state of the machine. For the most part, your program communicates with the operating system routines through structures that both the system and your program know about.

Typedefs: In the include file "include/exec/types.h", there are a bunch of typedef statements that define commonly used variable types. The typedef statements have two purposes: To make your program more portable, and for convenience. Here are two examples of typedefs, taken from the "include/exec/types.h" include file:

```
typedef short SHORT;
typedef unsigned char *STRPTR;
```

If you define a variable as SHORT (16 bits), and you wish to later use a compiler that interprets a short int as 8 bits, for example, you can just change the typedef statement instead of changing all of your variable declarations. The second typedef above can help to make your programs a bit shorter and clearer, as you can define pointers to strings with:

```
STRPTR x, y;
```

instead of having to say:

```
unsigned char *x, *y;
```

Comments: The Amiga's include files are full of comments that explain all of the structures and in many cases, how to use them. Much valuable information can be gained from reading the include files. To save space on your program development disk, you can strip these comments out (there are public domain programs available to do this), but you should keep copies of the original files handy for reference – they can sometimes clear up things that are confusing in the manuals.

Libraries

The term "library" can be confusing on the Amiga, as there are two quite distinct kinds. There are the standard libraries that you link with, as you would in any C or assembler program. For example, you link your compiled C program with such a library to include any standard functions like printf, strcmp, etc. With Amiga's Lattice C compiler, you must link with "amiga.lib" to use library functions; the Manx Aztec C equivalent is called "c.lib". This sense of the word 'library' should be nothing new for C or assembler programmers coming from other environments.

But just to throw you off (c'mon, you don't want this to be *too* easy, do you?), there is another kind of library in the Amiga. To access any of the system routines, you have to open a library. Several of these libraries are available, each one covering a different aspect of operating system. An example is the graphics library, which is in ROM (actually, the writable control store, but we'll stick to the convention and call it ROM), and contains all the graphics primitives for drawing lines, plotting text, filling areas, etc. There are over a dozen libraries like this available to the Amiga programmer; some are in ROM and some are on the WorkBench disk in the "libs" directory, but they are all used in the same way. (If a program requests a library that is not in ROM, the system will ask you to insert the disk you booted with if it isn't already in the drive.)

To use any of the system's routines, you have to "open" the library in which that routine is found. Opening the library determines its "library base", which is the place in memory where a jump table exists for all the routines in the library. (Library bases are not constant: disk-based libraries can be anywhere in RAM after they're brought in, and ROM-based libraries can move around with different operating system releases.) You never have to worry about the actual location of the library base: in C, you just open the library you want using the 'OpenLibrary' function, then call the routines in it just as you would call any other function. In assembler, it's a bit more complicated, and we'll postpone discussion of that till a bit later on.

First, let's look into how the library routines are actually executed. You don't need to know all of this stuff in order to use system routines, but most programmers prefer to know what's actually going on instead of just using a magic incantation because it works.

When you call OpenLibrary, you give it the name of the library you want to open (e.g. "graphics.library"), and the version number of the operating system that your program needs (or zero if it doesn't matter); it gives you back a pointer to the library base. The system can call any routine in that library by using offsets from the library base. Where does it get the values of those offsets? They're defined in the library with which you link ("amiga.lib" with the standard assembler and C compiler). In memory locations below the library base, there is a jump table for all of the routines in the library, just like the Kernal jump table in the 8-bit Commodore machines. Above the library base is a structure that holds global data needed by routines in the library.

Now you may have noticed that we've pulled a fast one on you. We've said that in order to use any system routine, you need to

open the library containing it. We've also said that in order to open a library, you use the OpenLibrary routine. "Hmmm", you wonder, "how do you open the library containing the OpenLibrary routine?" Good question. OpenLibrary is contained in "exec.library", which you can find by relying on the only fixed memory location in the entire system. Location 000004 holds a pointer to "ExecBase", the library base for the Exec library.

When you program in C, the Exec library is automatically opened for you (as is the DOS Library), but in assembler you have to use location 4 to get the pointer to ExecBase yourself. Once you have this pointer, you can use the offset (whose value will be defined when you link) to open the library. For details about using system routines in assembler and some sample code, see the article "An Introduction to Machine Language Programming on the Amiga" in this issue. It covers that topic nicely, so we'll concentrate a bit more on C right now.

When you get the library base pointer from the OpenLibrary function, you must assign it to a specific variable name so that the routines in "amiga.lib" know the library base. Your program can also use this variable, since it also points to the library structure. (Remember, the jump table for the routines is *below* the library base, the library structure is above.) The names of the library base variables for some of the libraries are:

Library	Library Base Name
exec.library	ExecBase
dos.library	DosBase
intuition.library	IntuitionBase
graphics.library	GfxBase
layers.library	LayersBase
clist.library	ClistBase
mathffp.library	MathBase
translator.library	TranslatorBase

Since the library base is used as a pointer to the library structure, you must declare it as such. (The library base structure template, has been defined, as usual, in an include file you used in your program.) The OpenLibrary function should be declared as returning a generic pointer, since it is used to open all kinds of libraries, with their own structure templates. Since the library base variable has been declared as a pointer to the library structure, you should 'cast' the OpenLibrary function in the assignment. The following C program shows how to open a library, in this case the intuition library.

```
#include <intuition/intuition.h> /* include for intuition library */
#include <exec/types.h>          /* type definitions, etc.       */

/* declare OpenLibrary as function returning a pointer */
/* (APTR is defined in "exec/types.h" as a generic pointer)     */
APTR OpenLibrary( );

/* declare library base as a pointer to library structure */
struct IntuitionBase *IntuitionBase;

main ( )

{
    /* open the library and cast result to appropriate type */
    IntuitionBase = (struct IntuitionBase *)
                    OpenLibrary("intuition.library", 0L);

    /* check if OpenLibrary failed for some reason */
    if (IntuitionBase = = NULL)
        exit(0L); /* (here we just bomb on failure) */

    DoStuff( ); /* your routine using calls to intuition routines */

    /* you should close all libraries you opened */
    CloseLibrary(IntuitionBase);
}
```

Closing the library isn't strictly necessary for ROM-based libraries, but if the library is WorkBench disk-based, closing it will tell the system that it can kick the library out if it needs the RAM space. Since all libraries are created equal, and you don't want to worry about which are ROM and which are disk-based, your programs should always close all libraries they have opened.

For a more complete look at how system routines are used, you should have a look at the source of a real program. Which brings us, conveniently, to the next section.

Colours: A Sample Program

Rather than give a simple "Hello World" example to show programming concepts, we decided to include here a real C program – something that is actually useful in itself and not just for instruction. To that end, the program "Colours" is presented here. Colours opens a draggable window with depth-arranging gadgets on the WorkBench screen. Proportional gadgets on the window let you change the Red, Green and Blue components of any colour register, as in the Preferences program. You can switch to higher or lower colour registers by clicking up/down arrow-gadgets. This in itself is fairly handy, but Colour's greatest attribute is that it can operate on the colour registers of another screen. It operates on either the frontmost screen or the second screen from the front, depending on your selection (click the right mouse button to select one or the other). This gives you the superb luxury of changing the colours of just about any program currently running in the system. For example, this article is being written on TextCraft, which has no provision for changing any of the eight colours it uses for various parts of the display. I currently have a purple background, aqua page, orange top margin and other colours that the people who wrote TextCraft probably only had nightmares about. To get these colours, I just brought the WorkBench screen to the front (using left-Amiga/N), where Colours was running (it uses no CPU time when not in use, so you can bring it up and forget about it). I then partially slid down the Workbench screen, revealing TextCraft's screen. From there, I just used Colours – set to "second screen" mode – and happily

mucked about with each of the eight colour registers, sliding the RGB gadgets until I saw a colour I liked. The actual value of the colour also appears on the Colours window, so you can reproduce any of the colours again, or use them in your programs. Most programs, including commercial games, can have their colours changed on them – right from under their nose! Great fun. The only programs you can't do that with are the ones that completely hog the system and don't let you get to other screens as long as the program is up, but those are no fun anyway and don't really belong on the Amiga.

Other features of Colours: You can only select "second screen" if there is one; pressing and holding an arrow gadget down for about half a second makes it repeat rapidly, letting you quickly get to any colour register (useful in 5-bitplane screens, which have 32 colour registers); the number of colour registers you can select always corresponds to how many are in the screen currently being operated on; if you're changing the colours of the screen containing the colours program, a small block is rendered in the current colour being changed; clicking anywhere on the window outside of a gadget moves the RGB gadgets to their "dead-on" positions for the current colour value.

The program uses the Intuition library extensively, using a window, proportional and boolean gadgets, and the IDCMP (Intuition Direct Communication Message Port) for receiving gadget and mouse button events. The colours are set by the Graphic Library's SetRGB4( ) function, and the Move( ) and Text( ) functions are used to place various labels about the window. The Intuition and Graphics libraries are the only ones explicitly opened by this program.

This program should show you that it's not too hard to use the Amiga's routines, Intuition's gadgets, and change system parameters like colours. And as a bonus, this is one of the few programs around that actually lets you change something on somebody else's screen! Some may consider that rude, but, at least in this case, it sure is handy!

Amiga System Structures

Whenever you write any Amiga-specific code, you'll find yourself using structures extensively to communicate with the system's library routines. In a way, structures on the Amiga are analogous to important system locations (such as those in zero-page) on a simpler computer like a Commodore 64. Anything you need to find out about the system, like colours of screens, sizes of windows, graphics modes, etc. can be found in a structure somewhere. Structures also contain pointers to other structures, so you can get to just about anything by using a bit of indirection. A good example of this can be found in the Colours program listed at the end of this article.

A structure is associated with each screen in the system. To get a pointer to the 'Screen' structure for the frontmost screen, we look in the 'IntuitionBase' structure, which we got a pointer to when we opened the Intuition library. The screen pointer is assigned to a variable called 'TheScreen', like this:

```
TheScreen = IntuitionBase->FirstScreen;
```

From a known pointer to a structure, we got a pointer to another structure of interest. Now, the program needs to know how many colour registers are in the first screen. Well, the Screen structure contains a pointer to a 'ViewPort' structure, which is a lower level representation of a Screen (a Screen is an Intuition entity, whereas a ViewPort is used by the lower level graphics routines that Intuition itself calls). The ViewPort doesn't directly provide what we're looking for, but does contain another structure called a 'RasInfo' structure – containing information about a raster (all bitplanes making up the display of a ViewPort). Inside the RasInfo structure is a pointer to a 'BitMap' structure, which finally contains almost what we're looking for – a member called 'Depth'. Depth is the number of bitplanes of the screen, and we want the number of colour registers, so we compute 2↑Depth and we're through. All of the above is done with the C assignment:

```
numregs = 1 << ((TheScreen->ViewPort)
                .RasInfo->BitMap->Depth);
```

The point is that you have to know what structures contain what, and how to get from one structure to another. The include files define the structures, so looking at the right include file is all you need to do to find out about any structure. The key is knowing what include file to look at, and it's not always obvious, since files include other files, which in turn may include others. What everyone really needs is a list of all include files, telling which files each includes and what structures each defines. And that, gentle reader, is exactly what we've prepared for you.

Directories: intuition
exec
workbench
graphics
libraries
resources
hardware
devices

FILE intuition/intuitionbase.h

```
#include "exec/libraries.h"
#include "graphics/view.h"

struct IntuitionBase
```

FILE intuition/intuition.h

```
#include "intuition/intuitionbase.h"
#include "graphics/gfx.h"
#include "graphics/clip.h"
#include "graphics/view.h"
#include "graphics/rastport.h"
#include "graphics/layers.h"
#include "graphics/text.h"
#include "exec/ports.h"
#include "devices/timer.h"
#include "devices/inputevent.h"

struct Menu
struct MenuItem
struct Requester
struct Gadget
struct PropInfo
struct StringInfo
struct IntuiText
struct Border
struct Image
struct IntuiMessage
struct Window
struct NewWindow
struct Screen
struct NewScreen
struct Preferences
struct Remember
```

FILE exec/exec.h

```
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/interrupts.h"
#include "exec/memory.h"
#include "exec/tasks.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/devices.h"
#include "exec/io.h"
```

FILE exec/libraries.h

```
#include "exec/types.h"
#include "exec/nodes.h"

extern struct Library
```

FILE exec/execbase.h

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/tasks.h"
#include "exec/libraries.h"
#include "exec/interrupts.h"

struct ExecBase

FILE exec/tasks.h

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"

extern struct Task

FILE exec/interrupts.h

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"

struct Interrupt
struct IntVector
struct SoftIntList

FILE exec/execname.h

No includes or structures in this file.

FILE exec/devices.h

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/tasks.h"
#include "exec/ports.h"
#include "exec/libraries.h"

struct Device
struct Unit

FILE exec/errors.h

No includes or structures in this file.

FILE exec/alerts.h

No includes or structures in this file.

FILE exec/resident.h

#include "exec/types.h"
#include "exec/nodes.h"

struct Resident

FILE exec/nodes.h

#include "exec/types.h"

struct Node

FILE exec/lists.h

#include "exec/types.h"
#include "exec/nodes.h"

struct List

FILE exec/io.h

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/tasks.h"
#include "exec/ports.h"

struct IORequest
struct IOStdReq

FILE exec/ports.h

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/tasks.h"

struct MsgPort
struct Message
struct Semaphore

FILE exec/types.h

No includes or structures in this file.

FILE exec/memory.h

#include "exec/types.h"
#include "exec/nodes.h"

struct MemChunk
struct MemHeader
struct MemEntry
struct MemList

FILE workbench/startup.h

#include "exec/types.h"
#include "exec/ports.h"
#include "libraries/dos.h"

struct WBStartup
struct WBArg

FILE workbench/icon.h

No includes or structures in this file.

FILE workbench/workbench.h

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/tasks.h"
#include "intuition/intuition.h"

struct DrawerData
struct DiskObject
struct FreeList
struct WBObject

FILE graphics/text.h

#include "exec/ports.h"

struct TextAttr
struct TextFont

FILE graphics/clip.h

#include <graphics/gfx.h>
#include <exec/ports.h>

struct Layer
struct ClipRect

FILE graphics/regions.h

#include <graphics/gfx.h>

struct RegionRectangle
struct Region

FILE graphics/gfxbase.h

#include <exec/lists.h>
#include <exec/libraries.h>
#include <exec/interrupts.h>

struct GfxBase

FILE graphics/copper.h

struct CopIns
struct cprlist
struct CopList
struct UCopList
struct copinit

FILE graphics/display.h

No includes or structures in this file.

FILE graphics/sprite.h

struct SimpleSprite

FILE graphics/view.h

#include <graphics/gfx.h>

struct ColorMap
struct ViewPort
struct View
struct RasInfo

FILE graphics/rastport.h

#include <graphics/gfx.h>

struct AreaInfo
struct TmpRas
struct GelsInfo
struct RastPort

FILE graphics/gfxmacros.h

#include <graphics/rastport.h>

FILE graphics/layers.h

#include <exec/ports.h>
#include <exec/lists.h>

struct Layer__Info

FILE graphics/gfx.h

struct Rectangle
struct BitMap

FILE graphics/graphint.h

#include <exec/nodes.h>

struct Isrvstr

FILE graphics/gels.h

struct VSprite
struct Bob
struct AnimComp
struct AnimOb
struct DBufPacket
struct collTable

FILE graphics/collide.h

No includes or structures in this file.

FILE libraries/mathffp.h

No includes or structures in this file.

FILE libraries/dos.h

#include "exec/types.h"

struct DateStamp
struct FileInfoBlock
struct InfoData

FILE libraries/dosextens.h

#include "exec/ports.h"
#include "exec/libraries.h"
#include "libraries/dos.h"

struct Process
struct FileHandle
struct DosPacket
struct StandardPacket
struct DosLibrary
struct RootNode
struct DosInfo
struct CommandLineInterface
struct DeviceList
struct FileLock

FILE libraries/diskfont.h

#include "exec/nodes.h"
#include "exec/lists.h"
#include "graphics/text.h"

struct FontContents
struct FontContentsHeader
struct DiskFontHeader
struct AvailFonts
struct AvailFontsHeader

FILE libraries/translator.h

No includes or structures in this file.

FILE resources/potgo.h

No includes or structures in this file.

FILE resources/cia.h

No includes or structures in this file.

FILE resources/misc.h

#include "exec/libraries.h"

struct MiscResource

FILE resources/disk.h

#include "exec/types.h"
#include "exec/lists.h"
#include "exec/ports.h"
#include "exec/interrupts.h"
#include "exec/libraries.h"

struct DiscResourceUnit
struct DiscResource

FILE hardware/custom.h

struct Custom

FILE hardware/cia.h

struct CIA

FILE hardware/dmabits.h

No includes or structures in this file.

FILE hardware/blit.h

struct bltnode

FILE hardware/adkbits.h

No includes or structures in this file.

FILE hardware/intbits.h

No includes or structures in this file.

FILE devices/narrator.h

#include "exec/io.h"

struct narrator_rb
struct mouth_rb

FILE devices/keymap.h

struct KeyMap

FILE devices/timer.h

#include "exec/io.h"

struct timeval
struct timerequest

FILE devices/printer.h

#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/ports.h"

struct IOPrtCmdReq
struct IODRPReq

FILE devices/serial.h

#include "exec/io.h"

struct IOTArray
struct IOExtSer

FILE devices/console.h

#include "exec/io.h"

FILE devices/gameport.h

struct GamePortTrigger

FILE devices/audio.h

#include "exec/io.h"

struct IOAudio

FILE devices/keyboard.h

#include "exec/io.h"

FILE devices/clipboard.h

#include "exec/ports.h"

struct ClipboardUnitPartial
struct IOClipReq
struct SatisfyMsg

FILE devices/input.h

#include "exec/io.h"

FILE devices/parallel.h

#include "exec/io.h"

struct IOPArray
struct IOExtPar

FILE devices/trackdisk.h

#include "exec/io.h"

struct IOExtTD

FILE devices/bootblock.h

struct BootBlock

FILE devices/prtbase.h

#include "exec/ports.h"
#include "exec/libraries.h"
#include "devices/parallel.h"
#include "devices/serial.h"
#include "devices/timer.h"
#include "libraries/dosextens.h"
#include "intuition/intuition.h"

struct DeviceData
struct PrinterData
struct PrinterExtendedData
struct PrinterSegment

FILE devices/inputevent.h

#include "devices/timer.h"

struct InputEvent

FILE devices/conunit.h

#include "exec/ports.h"
#include "devices/console.h"
#include "devices/keymap.h"
#include "devices/inputevent.h"

struct ConUnit

```
/*  >> Colours: The amazing colour-set control panel <<
**  >>              From The Transactor              <<
**
**   October 1986, Version 1.0
**
** - uses proportional gadgets for setting R G B
**   and allows alteration of any colour register
** - operates on the TOPMOST OR SECOND SCREEN, so
**   lets you change the colours of any currently
**   executing program with its own screen
**
**        >> THIS PROGRAM MAY BE FREELY DISTRIBUTED <<
**
** (c) 1986, AHA! (Acme Heuristic Applications!)
**
** Written by Chris Zamara and Nick Sullivan
*/

#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gfx.h>
#include <graphics/view.h>

/* defines for defaults, positioning, sizes, etc.       */
#define SCREEN2INIT FALSE /* default to topmost screen  */
#define GADGHEIGHT 10L    /* height of prop. RGB gadgets*/
#define RTOP 15L          /* y pos of top of Red gadget */
#define GTOP 28L          /* " " " Green gadget         */
#define BTOP 41L          /* " " " Blue gadget          */
#define CVAL 177L         /* x pos of colour values     */
#define CPOSX 195L        /* x pos of colr reg indicator*/
#define CPOSY 63L         /* y pos of " " ...           */
#define RPOSX 115L        /* x pos of Register text     */
#define SCRPOSX 15L       /* x pos of screen indicator  */
#define SCRPOSY 63L       /* y pos of screen indicator  */
#define UPGADGX 197L      /* x pos of up gadget         */
#define UPGADGY 10L       /* y pos of up gadget         */
#define DNGADGX 197L      /* x pos of down gadget       */
#define DNGADGY 32L       /* y pos of down gadget       */
#define COLORGADGET 1     /* ID for RGB gadgets         */
#define UPGADGET 2        /* ID for up gadget           */
#define DOWNGADGET 3      /* ID for down gadget         */
#define ARROWDELAY 5      /* # of ticks before repeat   */
#define POTINC 0x1111     /* amt added to pot per click */
#define RTEXT (RTOP+(GADGHEIGHT >> 1)+2) /* gadget      */
#define GTEXT (GTOP+(GADGHEIGHT >> 1)+2) /*   text      */
#define BTEXT (BTOP+(GADGHEIGHT >> 1)+2) /*   position  */

/* global structure declarations */
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct ViewPort *vp;
struct RastPort *rp;
struct IntuiMessage *GetMsg(), *message;
struct Window *OpenWindow(), *gwind;
struct Screen *TheScreen;

/* function declarations */
APTR   OpenLibrary();
USHORT GetRGB4();
USHORT GadgPressed();
BOOL   GadgReleased();

/* co-ordinate of points of up/down arrows */
SHORT UpXY[8]   = {13,0, 25,17, 0,17, 13,0};
SHORT DownXY[8] = {0,0, 25,0, 13,17, 0,0};

/* control knob Image structures */
struct Image knobR, knobG, knobB;

/* PropInfo structures for control gadgets */
struct PropInfo propinfR, propinfG, propinfB;

/* Border structure for Up-Arrow gadget */
struct Border UpArrow = {
   3, 2,
   1, 0, JAM1,
   4,
   UpXY,
   NULL
};

/* Border structure for Down-Arrow gadget */
struct Border DownArrow = {
   3, 2,
   1, 0, JAM1,
   4,
   DownXY,
   NULL
};

/* gadget structure for colour controls */
struct Gadget gadgR = {
   NULL,          /* pointer to next gadget (set later)  */
   16, RTOP, 155, GADGHEIGHT,   /* left/top/width/height*/
   GADGHNONE | GADGIMAGE,
   GADGIMMEDIATE | RELVERIFY,
   PROPGADGET,
   (APTR)&knobR,      /* rendering (for autoknob)     */
   NULL,              /* no select rendering          */
   NULL,              /* intuitext ptr (none used)    */
   NULL,              /* mutual exclude               */
   (APTR)&propinfR,   /* ptr to propinfo structure    */
   COLORGADGET,       /* ID used to check gadget type */
   NULL
};
struct Gadget gadgG, gadgB;

/* gadget structure for up/down colour reg select arrows*/
struct Gadget gadgUp = {
   NULL,
   UPGADGX, UPGADGY, 31, 21,
   GADGHCOMP,
   GADGIMMEDIATE | RELVERIFY,
   BOOLGADGET,
   (APTR)&UpArrow,
   NULL, NULL, NULL, NULL,
   UPGADGET,      /* gadget ID */
   NULL
};
struct Gadget gadgDown;

/* define NewWindow structure for our window */
struct NewWindow GadgWind = {
   100, 50, 233, 72,   /* left, top, width, height   */
   -1, -1,             /* use screen colours         */
   GADGETDOWN          /* IDCMP flags                */
      | GADGETUP
      | MOUSEBUTTONS
      | INTUITICKS
      | CLOSEWINDOW,
   WINDOWDEPTH         /* window flags               */
      | WINDOWCLOSE
      | WINDOWDRAG
      | RMBTRAP
      | ACTIVATE
      | SMART_REFRESH,
   &gadgUp,            /* first gadget in list       */
   NULL,
   (UBYTE *)"Colour Control 1.0", /* window title    */
   NULL,                          /* ptr to screen   */
   NULL,
   0, 0, 0, 0,      /* sizing limits (non-resizable) */
   WBENCHSCREEN
};

/* array for fast'n'easy int to ASCII conversion     */
char *cnum[32] = { "00", "01", "02", "03", "04", "05", "06", "07",
        "08", "09", "10", "11", "12", "13", "14", "15",
        "16", "17", "18", "19", "20", "21", "22", "23",
        "24", "25", "26", "27", "28", "29", "30", "31",
      };

USHORT colreg = 0;      /* current colour register */
BOOL Screen2Flag = SCREEN2INIT; /* f=top scr, t=2nd scr */
BOOL Old2Flag    = !SCREEN2INIT; /* Screen2Flag prev val */


main (argc, argv)

int argc;
char *argv[];
 {

 ULONG  msgclass;       /* message class from IDCMP      */
 USHORT msgcode;        /* message code from IDCMP       */
 APTR   IAddr;          /* pointer to gadget from IDCMP  */
 USHORT WhichGadg;      /* ID of selected gadget         */
 USHORT TickStart;      /* arrow repeat delay countdown  */
 BOOL   GadgSel = FALSE;/* t = colour gadget selected    */
 BOOL   RegSel = FALSE; /* t = up or down gadget selected*/
 BOOL   window_still_open = TRUE;

   /* open intuition and graphics libraries   */
   if ( (IntuitionBase = (struct IntuitionBase *)
         OpenLibrary ("intuition.library", 0L) ) == NULL)
      exit (0L);
```

```
if ( (GfxBase = (struct GfxBase *)
      OpenLibrary ("graphics.library", 0L) ) == NULL) {
   CloseLibrary (IntuitionBase);
   exit (0L);
}

/* second arg specifies default colour register */
if (argc == 2)
   colreg = atoi(argv[1]);

/* fill in structures */
propinfR.Flags = FREEHORIZ | AUTOKNOB;
propinfR.HorizBody = 1 << 12; /* body increment 1/16 */
/* propinf structures for R G B gadgets all alike    */
propinfG = propinfR;
propinfB = propinfR;

/* G and B gadgets same as R */
gadgG = gadgR;
gadgB = gadgR;
/* except for... */
gadgG.TopEdge = GTOP;
gadgB.TopEdge = BTOP;
gadgG.GadgetRender = (APTR)&knobG;
gadgB.GadgetRender = (APTR)&knobB;
gadgG.SpecialInfo = (APTR)&propinfG;
gadgB.SpecialInfo = (APTR)&propinfB;

/* define down gadget in terms of up gadget */
gadgDown = gadgUp;
gadgDown.GadgetRender = (APTR)&DownArrow;
gadgDown.LeftEdge = DNGADGX;
gadgDown.TopEdge  = DNGADGY;
gadgDown.GadgetID = DOWNGADGET;

/* link all gadgets by pointers */
gadgUp.NextGadget = &gadgDown;
gadgDown.NextGadget = &gadgR;
gadgR.NextGadget = &gadgG;
gadgG.NextGadget = &gadgB;

/* now try to open the window containing the gadgets */
if ((gwind = OpenWindow(&GadgWind)) == NULL) {
   CloseLibrary(GfxBase);
   CloseLibrary(IntuitionBase);
   exit(0L);
}

/* get ptr to rastport for graphics routines     */
rp = gwind->RPort;

/* set RGB gadget positions to current colours   */
SetColrs();

/* put up labels on window display               */
WinText();

/* put up colour register number indicator       */
RegIndicate();

/*** main event loop ***/
while (window_still_open) {

   /* wait politely, unless sliding a colour gadget */
   if (! GadgSel)
      Wait (1L << gwind->UserPort->mp_SigBit);

  while (message = GetMsg(gwind->UserPort)) {
   /* get what we need from the message port */
   msgclass = message->Class;
   msgcode  = message->Code;
   IAddr    = message->IAddress;
   ReplyMsg(message); /* reply to msg right away */

   /* now we can interpret the message */

   /* check for gadget selected */
   if (msgclass == GADGETDOWN)
      WhichGadg = GadgPressed(IAddr, &GadgSel,
                &RegSel, &TickStart);

   /* check if gadget released, even if off gadget*/
   else if ( (msgclass == MOUSEBUTTONS
         && msgcode == SELECTUP)
        || msgclass == GADGETUP )
      GadgSel =
            GadgReleased(IAddr, &RegSel, WhichGadg);

   /* alternate affected screen if r. button down */
   else if (msgclass == MOUSEBUTTONS) {
      if (msgcode == MENUDOWN) {
         Screen2Flag = !Screen2Flag;

            UpdateColReg(0);
         }
         /* update display if left button pressed */
         else if (msgcode == SELECTDOWN)
            UpdateColReg(0);
      }

      /* auto-repeat arrow gadgets after delay */
      else if (msgclass == INTUITICKS && RegSel) {
         /* wait ARROWDELAY ticks before repeating */
         if (TickStart++ > ARROWDELAY)
            UpdateColReg(WhichGadg);
      }

      /* finish up if the close gadget is clicked */
      else if (msgclass == CLOSEWINDOW)
         window_still_open = FALSE;

      }
         if (GadgSel)
            /* set colour to pot values */
            ReadGadg();

      }

      /* close everything up before ending */
      CloseWindow(gwind);
      CloseLibrary(GfxBase);
      CloseLibrary(IntuitionBase);
}


USHORT GadgPressed (IAddr, GadgSel, RegSel, TickStart)
/* set flags, call right routines when gadget clicked */

struct Gadget *IAddr;
BOOL   *GadgSel, *RegSel;
USHORT *TickStart;
{
USHORT WhichGadg;

   /* get gadget type */
   WhichGadg = IAddr->GadgetID;

   /* rgb gadget? */
   if (WhichGadg == COLORGADGET) {
      *GadgSel = TRUE;
      /* update settings if 2nd screen gone */
      if (Screen2Flag &&
          IntuitionBase->FirstScreen->NextScreen == NULL)
         UpdateColReg(0);
   }

   /* else, must be up/down gadget */

   else {
      *RegSel = TRUE;
      *TickStart = 0; /* start delay timer */
      UpdateColReg(WhichGadg);
   }

   return (WhichGadg);
}


BOOL GadgReleased (IAddr, RegSel, WhichGadg)
/* gadget released - set flags, call relevant routines */

struct Gadget *IAddr;
BOOL   *RegSel;
USHORT WhichGadg;
{
BOOL GadgSel = TRUE;

   if (WhichGadg == COLORGADGET) {
      ReadGadg();
      if ( ( ((struct PropInfo *)
            (IAddr->SpecialInfo))->Flags ) & KNOBHIT )
         SetColrs(); /* reposition knob if it was moved */

      GadgSel = FALSE; /* colour gadget no longer sel.  */
   }
   else
      *RegSel = FALSE; /* other gadget no longer sel.   */
```

```
    return (GadgSel);
}


ReadGadg ()
/* read pot values, set register 'colreg' accordingly */

{
USHORT RedVal, GrnVal, BluVal;

    RedVal = propinfR.HorizPot / POTINC;
    GrnVal = propinfG.HorizPot / POTINC;
    BluVal = propinfB.HorizPot / POTINC;

    /* get viewport of topmost or second screen */
    vp = &(TheScreen->ViewPort);

    /* change colour register */
    SetRGB4(vp, (ULONG)colreg, (ULONG)RedVal,
            (ULONG)GrnVal, (ULONG)BluVal
            );

    PrintValues(RedVal, GrnVal, BluVal); /* nums to right*/
}

PrintValues (R, G, B)
/* print colour values to right of gadgets */

USHORT R, G, B;
{
    SetAPen(rp, 1L);
    Move(rp, CVAL, RTEXT);
    Text(rp, cnum[R], 2L);
    Move(rp, CVAL, GTEXT);
    Text(rp, cnum[G], 2L);
    Move(rp, CVAL, BTEXT);
    Text(rp, cnum[B], 2L);
}


SetColrs ()
/* set gadget pot values according to rgb in 'colreg' */

{
USHORT CO, R, G, B;
ULONG hpR, hpG, hpB;

    /* get viewport of topmost screen */
    ScreenPick(); /* select screen one or two */
    vp = &(TheScreen->ViewPort);

    /* get rgb of specified colour register */
    CO = GetRGB4(vp->ColorMap, (ULONG)(colreg) );

    /* convert rgb to HorizPot values */
    R = (CO >> 8) & 0xF;
    G = (CO >> 4) & 0xF;
    B = CO & 0xF;
    hpR = R * POTINC;
    hpG = G * POTINC;
    hpB = B * POTINC;

    /* change gadgets to reflect new colours */
    ModifyProp(&gadgR, gwind, NULL,
          (ULONG)propinfR.Flags, hpR, 0L,
          (ULONG)propinfR.HorizBody, 0L
         );
    ModifyProp(&gadgG, gwind, NULL,
          (ULONG)propinfG.Flags, hpG, 0L,
          (ULONG)propinfG.HorizBody, 0L
         );
    ModifyProp(&gadgB, gwind, NULL,
          (ULONG)propinfB.Flags, hpB, 0L,
          (ULONG)propinfB.HorizBody, 0L
         );

    PrintValues(R, G, B); /* colour numbers */
}

RegIndicate ()
/* show colour register number */

{
ULONG c;

    /* first print register number */
    Move(rp, CPOSX, CPOSY);
    SetAPen(rp, 1L);
    Text(rp, cnum[colreg], 2L);
    /* show a block in 'colreg' colour if this screen */
    /* or background colour if another screen         */
    c = (IntuitionBase->ActiveScreen == TheScreen)
          ? colreg : 0;
    Move(rp, CPOSX+24, CPOSY);    /* move past text     */
    SetDrMd(rp, INVERSVID);       /* inverse mode       */
    SetAPen(rp, c);               /* set colour         */
    Text(rp, " ", 1L);            /* print a space      */
    SetDrMd(rp, JAM2);            /* set mode to normal */
    SetAPen(rp, 1L);
}

WinText ()
/* put up various labels on window */

{
    SetAPen(rp, 1L);

    Move(rp, 5L, RTEXT);
    Text(rp, "R", 1L);

    Move(rp, 5L, GTEXT);
    Text(rp, "G", 1L);

    Move(rp, 5L, BTEXT);
    Text(rp, "B", 1L);

    Move(rp, RPOSX, CPOSY);
    Text(rp, " Register", 9L);

}


UpdateColReg (WhichGadg)
/* increment or decrement colour register          */
/* if WhichGadg is 0, just display current settings */

USHORT WhichGadg;

{
USHORT numregs;

    /* get number of bitplanes of screen */
    ScreenPick(); /* set 'TheScreen' */
    numregs = 1 << ((TheScreen->ViewPort).RasInfo->
                BitMap->Depth);
    /* (whew!) */

    /* increment or decrement colour register */
    if (WhichGadg == UPGADGET)
       colreg++;
    else if (WhichGadg == DOWNGADGET)
       colreg--;

    colreg = colreg % numregs;

    RegIndicate();
    SetColrs();
}


ScreenPick ()
/* select top screen if Screen2Flag false  **
** or second screen if true                **
** and update display if necesary          **
*/

{
    TheScreen = IntuitionBase->FirstScreen;

    Move(rp, SCRPOSX, SCRPOSY);
    SetAPen(rp, 1L);
    SetBPen(rp, 3L);   /* print in new background colour */

    if (Screen2Flag) {
       if (TheScreen->NextScreen == NULL) /* no 2nd scrn */
          Screen2Flag = FALSE;
       else {
          /* set pointer to second screen */
          TheScreen = TheScreen->NextScreen;
          if (Old2Flag != Screen2Flag)
             Text(rp, "SECOND SCREEN", 13L);
       }
    }
    if (!Screen2Flag && Old2Flag)
       Text(rp, " TOP  SCREEN ", 13L);

    SetBPen(rp, 0L);
    Old2Flag = Screen2Flag;
}
```

A Tale Of Two Cs

Adam Herst
Toronto, Ontario

. . .the most recent programming revolution has made it to Commodore machines in a big way.

It is the best of times, it is the worst of times. The C–64, revolutionary in its time for its price/performance ratio, has been surpassed. While the C–128 improves on the C–64, it offers no technological advances and is in danger of being priced off the market. It often seems that the newest and brightest technologies and programs never make it to this part of the Commodore world. A good hard–disk drive, for instance, is sorely lacking. Nonetheless, the most recent programming revolution has made it to Commodore machines in a big way. The C programming language, often said to be the most–used language for large, complicated programming projects, is now available for many Commodore machines.

The reasons for this are two–fold. With the introduction of the C–128 and its very standard CP/M operating system, many tried and tested C compilers now run on a Commodore machine. The release of the Amiga has also seen the introduction of many C compilers for it. While requiring custom versions for its unique operating system, the Amiga's configuration of chips and OS make it an ideal C machine. Finally, the successful implementation of OS/9 on the SuperPET and the subsequent introduction of a full C package for it helped extend the Commodore C shores.

While the new Commodore machines have helped make C a viable alternative programming language for Commodore users, the second reason is the interest and support given the language by two software manufacturers. Abacus Software of Michigan and Pro–Line Software of Toronto have, in the last couple of years, introduced and supported versions of C for the C–64 and C–128. Reasonably priced and relatively standardized, C is now available for the most popular of Commodore machines.

Abacus software has been producing Super C for the C–64 for over two years. While a good first offering, the original versions suffered from poor organization, sparse function libraries and a non–standard implementation. A serious programming environment was not provided with the various components of the system (editor, compiler and linker) linked together by a simple loader menu. With the introduction of the C–128 came Super C version 3.0 for the C–128. Not content merely to port the compiler to the new machine, many of the problems in the original C–64 version have been addressed. Libraries have been dramatically improved and now include graphic and math functions. In addition, all of the component programs have been tied together under an operating system–like shell that provides greater system control. Perhaps the best news is that Super C for the C–64 has been upgraded to version 2.0 and includes the library and shell enhancements of the 128 version. Programs for the two versions are portable within machine restrictions.

Released at approximately the same time as Super C was C Power from Pro–Line Software. Generally agreed to be superior to Super C version 1.0, it included the extensive libraries and shell interface that have only just appeared in the Abacus offerings. In addition, the C Power implementation more closely adhered to accepted C standards, a must for a language which claims portability as one of its strong points. A C–128 version of C Power is now available, offering little change from the C–64 version other than machine–dependent improvements. (Unless otherwise noted, references to one or the other of the two compilers will be to both the C–64 and C–128 versions).

Documentation

Super C is clearly the winner here. A three–ringed binder with close to 300 pages of tutorial and reference material is included in the package. Divided into two sections, the first is a user guide and comprises tutorials on the various shell utilities and short sample programs to familiarize the novice with the operation of compilers and the peculiarities of the C language. The second section is a system guide and comprises an extensive reference to the shell utilities and Abacus's implementation of the language. Despite the volume of information, a much–improved index makes specific references easy to find.

C Power suffers from poor documentation. A short–coming of the C–64 version, it was alleviated somewhat by the inclusion of C Primer Plus, an excellent introductory textbook. The small amount of system specific information was contained in some 40 odd pages of very small type with no index provided. In the C–128 version of C Power, Pro–Line has not included C Primer Plus and has made only small improvements to its documentation. Grown to 60 pages, it now includes a number of examples and tutorials and a one page index. Despite these improvements it conveys nowhere near the amount of information as the Super C package.

The Environment

Both compilers make use of a shell to tie together the various components of the programming environment. This is the first program that must be loaded and in both cases, exiting the shell resets the computer. As well as the compiler–specific files (editor, compiler, linker) are a number of built–in and transient utilities for system control.

Command–line driven, the Super C shell (an unintentional pun) can only be described as an MS–DOS / CP/M mutant. Letters represent drives which are logged into the system through their

letter designations. Up to eight disk drives can be supported. Built–in commands include DOS, time of day clock and a processor speed toggle command (on the C–128 version only). Transient commands include device, copy, type and a fast load command for use with 1541s. While not a major criticism, there is no way to set the screen colours for the system; cursor and text colours have special meaning, reflecting the current status of the system. I was never able to figure out what all the different colours meant and would gladly have sacrificed this 'feature' for a blue screen with white letters. A major plus is the ability to handle multiple command lines. This allows the compilation and linking process to be totally automated. In addition, parameters can be passed to the called programs through the command line.

I have been told that the C Power interface closely resembles that of the UNIX operating system. If so, I can understand why I've never seen anyone using a UNIX machine. I never have and can't confirm any resemblance. All in all, I wasn't impressed with the C Power shell, finding it clumsy to use. Four drives are supported, numbered 0 through 3, with drive 0 fixed as the logged drive. While a drive path is searched on file load, most DOS commands are directed to this drive. Also missing are the benefits of Commodore's unsurpassed full–screen editing. Incorrect command lines must be retyped –– in full. No cursoring up to make corrections. There isn't even a way to recall the last command issued. Built–in commands include the standard DOS commands and a set colour command. Also provided is I/O redirection. Transient commands (written in C with source code included) are provided mainly to illustrate I/O redirection and include find, sort, and wordfreq.

Both systems use the C–128s extra memory as a RAM disk and provide different commands with which to interact with them. The benefits are two–fold. Single–drive users can avoid the interminable disk swapping compilation requires and all users can benefit from the vastly increased compile times. C Power provides 191 blocks shared between two disks, while Super C provides a single disk of 239 blocks. The former includes commands for enabling and disabling the disks, thereby freeing the memory for program use, while the latter includes commands for the backup of the RAM disk into a single file on a floppy. While Super C claims to support the 1750 memory expansion with a RAM disk of up to 256K it is, as yet, an unimplemented 'feature'.

The Editor

Both packages provide an editor with which to create C source code. Given the fact that both packages seize control of the machine, using a third party editor is impractical so a system editor is a must. However, as these are compiler and not editor packages, both leave a lot to be desired. Fundamental functions are provided. Both have adequate search & replace and cut & paste facilities. My biggest complaint here is that the cut and paste is line rather than character oriented. Super C provides 43k for text space that is divided between a main and an extra text area. C Power will give variable amounts of space depending on the existence of RAM disks and an unlimited number of editing buffers. Both provide the extra characters crucial to C (curly braces, backslash, vertical bar, tilde) although only the Super C editor will send the modified characters to your printer (even a CBM 1526!)

Two versions of the editor are provided with C power. One gives a larger text area while the second sacrifices editing space for syntax checking. This is a worthwhile trade–off especially when porting other source code. The code is loaded into the still full–featured editor, and syntax is checked when the command is issued. The process is halted on the first error and the cursor returns to command mode. Exiting command mode leaves the cursor on the source of the error. A very nice addition.

Both editors are adequate. Although Super C seems to have more bells and whistles, the C power editor has a much nicer 'feel' (a totally subjective opinion formed after using countless text editors and word–processors for innumerable hours.) Much of this opinion is due to the absence of an 'insert' mode of text entry in the Super C editor. This was compounded by a seemingly random bug that caused stray numbers to appear in the text. This occurred infrequently but did halt the compilation process a number of times with syntax errors.

A second drawback to the Super C editor is a fault of the system as a whole. To capitalize on the use of colour in the source code, the Super C editor and compiler use a USR file format. While the editor will load SEQ files (albeit in a garbled form), the compiler and linker will only accept USR files. This makes it very difficult to port foreign source code into the Super C system. File conversion is a must. As a solution I have prepared Convert, a program in C that will compile on both compilers. This program has also been used for the comparative benchmarks. It will convert Super C to sequential files and vice–versa.

Compiler

Both compilers support nearly standard implementations of the C language. Missing in both are bit fields, a means of assigning the number of bits that an object will require. It is unlikely that you will miss them. As well both have non–standard aspects in their initialization of variables and their definition of scope. This can become a problem when porting foreign code. Both compilers support all of the required data types. Super C holds the edge in type sizes with long int equal to 4 bytes compared to C Powers 2, and in precision with long float equal to 8 and 5 bytes respectively.

Both compilers generate error messages on compilation. Super C directs these to an error file which can be loaded into the extra text area of the editor for debugging. Extensive explanation of the error messages are contained in the documentation. Compilation continues until the end of the code or a fatal error has been reached. In either case, enough information has been accumulated to make major corrections to the source code. C Power will pause on encountering an error, and seems to give up on compilation much more readily. In addition, no error file is generated. These factors would be a major annoyance if not for the preventive error checking done by the C Power editor. Unfortunately, no documentation about the error messages is provided.

The compilation times for the Convert file are given below. All of the 128 processing is done in fast mode with a 1571 disk drive. The 128 versions allow compilation to disk, and this is reflected in their faster compilation times.

	To Disk	To RAM–Disk
Super C 64	42 s	NA
C Power 64	30 s	NA
Super C 128	20 s	4 s
C Power 128	88 s	50 s

Linker

The linker is used to attach library files to the compiled source code. Errors encountered during linking are written to the screen for both systems. Super C is able to link files from and to the RAM disk. Functions are contained in one of three library files, either stdio, math or graphics. In all cases the entire library is linked to the compiled source code. C Power, while claiming to allow linking from and to the RAM disk, was not above occasionally trashing my RAM disk during the process. Functions exist as independent files, and only referenced functions are linked to the compiled source. This can result in substantially smaller program code. The various link times are listed below.

	To Disk	To RAM–Disk
Super C 64	85 s	NA
C Power 64	77 s	NA
Super C 128	44 s	5 s
C Power 128	71 s	47 s

Libraries

The key to C's portability is its use of function libraries. While the heart of a C compiler should remain constant across implementations, I/O and machine specific functions are included by the manufacturer as external libraries. The contents of the library depend on the manufacturer and are one of the major determinations of a compiler's utility. A complete function library saves a programmer many hours of writing his/her own libraries. Key functions are outlined in the K&R standard for C and these are contained in both the Super C and the C Power compilers. In addition Super C provides extensive graphic and math functions not available in the C Power package.

Machine Language

A function provided by both compilers allows access to machine language functions. This is the call() function in Super C and the sys() function in C Power. Since I do not program in ML (one of my reasons for learning C) I was not able to test these functions. Superficially, the C Power ML interface appears superior. It allows the passing of bank, address and register parameters while the Super C call() function passes a single address pointer. As well, simple memory maps of the compiler environments are provided to help the programmer prevent fatal interaction between the C and ML code. Details on machine language access is the only area in which the C Power documentation surpasses that of Super C.

Run Time

In all but the C–128 version of Super C, the product of linking can be designated to run under the shell or as a stand–alone program. For both versions of C Power and Super C for the C–64, this stand alone–program will run from BASIC and can be freely distributed. The C–128 version of Super C requires that a special disk be prepared with the shell on it using the sysgen command. An autoexec file can be included to cause the program to auto–boot. Among other benefits, this allows the RAM disk to be distributed with the program. (It should be noted, however, that the licensing agreement prohibits the ". . .copy(ing) of any portion of this software package. . .for any purpose other than backup.". This industry is crazy.)

C Power holds the edge in program size. Resultant files are considerably smaller than those produced with Super C. This is due at least in part to the selective linking of library functions in C Power compared with the all or none linking of Super C. File sizes for the Convert program are provided below. Execution times were measured using the Convert program to convert a 100 block file. Sizes and times are presented below.

	Size	Run time
Super C 64	40 blocks	6:19
C Power 64	16 blocks	4:44
Super C 128	41 blocks	4:00
C Power 128	21 blocks	3:57

Conclusion

Both the Super C compiler from Abacus Software and the C Power compiler from Pro–Line are good implementations of the C language. Neither without fault, they have different qualities that recommend both for different needs. C Power's weak documentation cripples what would otherwise be a well rounded package. More examples of compiler–specific functions and error messages, and an expanded function library, are all that are needed. Super C's major weakness only applies to the C–128 version. The inability to create a run from BASIC program is one of the few features that would have better been left unchanged from the original C–64 version. Arnie –– get on the ball!

Each of the compilers fill a different need. For the novice and intermediate C programmer, Super C must be recommended on the strength of its documentation and the breadth of its libraries. It is an easy to use package with more power that most users should need. Its major faults are only apparent to the discerning eye. For the expert programmer or software developer the issue is a toss–up. On the 128, Super C's ultra–fast compile and link times make program development effortless. In addition there are a wide range of graphic and math functions to choose from. If execution speed or run from BASIC capacity is important, however, then C Power may be the way to go.

The reasons for programming in C are myriad. A fully structured language, C also provides extensive control over variables and functions. Combined with this are many of the advantages of programming in assembly language: fast execution and low–level control of the machine. While no language is ever the ultimate solution for all programming projects, C can fulfill important needs in the Commodore community. Its transportability ties together a wide range of Commodore computers in a way BASIC 2.0 was never able to. It is gratifying to see that the 65xx family of Commodore machines is not being left out of this most recent programming revolution.

```
/* program to convert Super C      */
/* files to/from SEQ files         */
/* converting filetypes            */
/* and control codes               */

#define POWER
#ifdef POWER
#include "stdio.h"
#define CLR '\223'
#else
#include "h:stdio.h"
#endif

char inname[25];
char outname[25];
#ifdef POWER
    FILE inchan;
    FILE outchan;
#else
file inchan;
file outchan;
#endif

main()
{
    int menu();

    int choice ;
    int status = 0;
    char inbuff[255];
    char outbuff[255];

    while ( ( choice = menu() ) != '0' )
    {
         choice = ( choice == '1' ) ? 1 : 2 ;
         putchar(CLR);
#ifdef POWER
             getchar();
#endif
         namefile("Source ", inname);
         namefile("Destination ", outname);
         modnames( choice );
#ifdef POWER
            inchan = fopen(inname,"r");
            outchan = fopen(outname,"w");
#else
            inchan = open(8,2,inname);
            outchan = open(8,3,outname);
#endif
status = 0;

         while (!status) /* read a line and convert it */
         {
             fgets(inbuff, 254, inchan);
#ifdef POWER
                status = feof(inchan);
#else
                status = EOF;
#endif
             convert(choice, inbuff, outbuff);
             fputs(outbuff, outchan);
             printf("\n%s", outbuff );
         }

         close(inchan);
         close(outchan);
    }
}

 modnames( choice )  /* modify filename for */
 int choice;
 {                   /* read/write usr/seq  */
    switch (choice)
    {
        case 1 :
            strcat(inname, ",u,r");
            strcat(outname, ",s,w");
            break;
        case 2 :
            strcat(inname, ",s,r");
            strcat(outname, ",u,w");
            break;
    }
 }

 namefile(filetype, name)                    /* input filename */
 char *filetype;
 char *name;                                 /* pass pointer to filename */
 {
    printf("\n%sfile name: ",filetype);
    gets(name,16);
                          /* Super C gets() fix */
    if ( *( name + ( strlen(name) -1 ) ) == '\n' )
      *( name + ( strlen(name) -1 ) ) = '\0' ;
 }

int menu()
{
    int choice;
    do
    {
        printf("%c\n",CLR);
        printf(" 1. Super C to Sequential\n");
        printf(" 2. Sequential to Super C\n\n");
        printf(" 0. EXIT");
        choice = getchar();
    }
    while ( ( choice != '0' ) && ( choice != '1' ) && ( choice != '2' ) );
    return choice;
}

convert(choice, inbuff, outbuff)
int choice;
char inbuff[];
char outbuff[];
{

   static int linenum = 0 ;
   int inndex=0;
   int outndex=0;

   if ( linenum == 0 )
       switch (choice)
       {
           case 1:
               inndex +=3 ;
               linenum = 1 ;
               copybuff(inbuff, outbuff, inndex, outndex);
               break;

           case 2:
               outbuff[0] = 133;
               outbuff[1] = 129;
               outbuff[2] = 5;
               outndex +=3;
               linenum = 1;
               copybuff( inbuff, outbuff, inndex, outndex );
               break;
       }
   else
       switch (choice)
       {
           case 1:
                inndex++;
                copybuff(inbuff, outbuff, inndex, outndex);
                break;

           case 2:
                   outbuff[0] = 5;
                   outndex++;
                   copybuff(inbuff, outbuff, inndex, outndex);
                   break;
       }
}

copybuff( inbuff, outbuff, inndex, outndex )
char inbuff[];
char outbuff[];
int  inndex;
int  outndex;
{
   do
   {
       outbuff[  outndex ] = inbuff[  inndex ];
       inndex++;
       outndex++;
   }
   while( inndex <= strlen( inbuff ) );
}
```

```
/* program to convert Super C      */
/* files to/from SEQ files         */
/* converting filetypes            */
/* and control codes               */

/* #define POWER    *** un-comment for C-Power */
#ifdef POWER
#include "stdio.h"
#define CLR '\223'
#else
#include "a:stdio.h"
#endif

char inname[25];
char outname[25];
#ifdef POWER
    FILE inchan;
    FILE outchan;
#else
file inchan;
file outchan;
#endif

main()
{
    int menu();

    int choice ;
    int status = 0;
    char inbuff[255];
    char outbuff[255];

    while ( ( choice = menu() ) != '0' )
    {
         choice = ( choice == '1' ) ? 1 : 2 ;
         putchar(CLR);
#ifdef POWER
             getchar();
#endif
         namefile("Source ", inname);
         namefile("Destination ", outname);
         modnames( choice );
#ifdef POWER
            inchan = fopen(inname,"r");
            outchan = fopen(outname,"w");
#else
            inchan = open(8,2,inname);
            outchan = open(8,3,outname);
#endif
status = 0;

         while (!status)   /* read line and convert it */
         {
             fgets(inbuff, 254, inchan);
#ifdef POWER
                 status = feof(inchan);
#else
                 status = EOF;
#endif
             convert(choice, inbuff, outbuff);
             fputs(outbuff, outchan);
             printf("\n%s", outbuff );
         }

         close(inchan);
         close(outchan);
    }
}

 modnames( choice )              /* modify filename for */
 int choice;
 {                               /* read/write usr/seq  */
    switch (choice)
    {
        case 1 :
            strcat(inname, ",u,r");
            strcat(outname, ",s,w");
            break;
        case 2 :
            strcat(inname, ",s,r");
            strcat(outname, ",u,w");
            break;
    }
 }

 namefile(filetype, name)  /* input filename            */
 char *filetype;
 char *name;               /* pass pointer to filename */
 {
    printf("\n%sfile name: ",filetype);
    gets(name,16);

    /* Super C gets() fix */
    if ( *( name + ( strlen(name) -1 ) ) == '\n' )
      *( name + ( strlen(name) -1 ) ) = '\0' ;
 }

int menu()
{
    int choice;
    do
    {
        printf("%c\n",CLR);
        printf(" 1. Super C to Sequential\n");
        printf(" 2. Sequential to Super C\n\n");
        printf(" 0. EXIT");
        choice = getchar();
    }
    while ( ( choice != '0' ) && ( choice != '1' )
           && ( choice != '2' ) );
    return choice;
}

convert(choice, inbuff, outbuff)
int choice;
char inbuff[];
char outbuff[];
{

   static int linenum = 0 ;
   int inndex=0;
   int outndex=0;

   if ( linenum == 0 )
       switch (choice)
       {
           case 1:
               inndex +=3 ;
               linenum = 1 ;
               copybuff(inbuff, outbuff, inndex, outndex);
               break;

           case 2:
               outbuff[0] = 133;
               outbuff[1] = 129;
               outbuff[2] = 5;
               outndex +=3;
               linenum = 1;
               copybuff(inbuff, outbuff, inndex, outndex);
               break;
       }
   else
       switch (choice)
       {
           case 1:
               inndex++;
               copybuff(inbuff, outbuff, inndex, outndex);
               break;

           case 2:
               outbuff[0] = 5;
               outndex++;
               copybuff(inbuff, outbuff, inndex, outndex);
               break;
       }
}

copybuff( inbuff, outbuff, inndex, outndex )
char inbuff[];
char outbuff[];
int  inndex;
       int  outndex;
       {
          do
          {
              outbuff[ outndex ] = inbuff[ inndex ];
              inndex++;
              outndex++;
          }
          while( inndex <= strlen( inbuff ) );
       }
```

A Comparison of Language Speeds

Anton Treuenfels, Fridley, Minnesota
Donald Piven, Chicago, Illinois
Brian Junker, Champaign–Urbana, Illinois

There are many reasons to consider using a programming language other than Basic on the C64.

(Note: these results were originally reported in a specialized "notesfile" devoted to Commodore on the Control Data PLATO system. PLATO is primarily devoted to computer–assisted instruction, but has also been an influential pioneer in such areas as multi–player interactive games and electronic "bulletin boards", things only recently rediscovered by organizations like CompuServe and The Source.)

There are many reasons to consider using a programming language other than Basic on the C64. Some of these reasons are: use of a structured programming language, easy use of a program originally written in another language, and taking advantage of features unique to a particular language. The most common reason, however, is speed. Basic is often perceived as simply not fast enough to suit some need.

Speed is so paramount a concern that sometimes the only question asked about a particular programming language (or a particular implementation of that language) is, "How fast is it?". Unfortunately, this is also a very difficult question to answer. A particular language may be well–suited to task X but not really meant for task Y, while for another language exactly the opposite may be true. A test of only task X or only task Y might not be very informative – particularly if a programmer's main interest is task Z.

Bearing that in mind, we present here the cheerfully unscientific methods and results of an informal survey to see how long it took to accomplish one particular task using several different languages and implementations of those languages on the C64. We also threw in a quick look at the Basic interpreters of the C16 and C128.

The task chosen was to determine the first 1000 prime numbers. Prime numbers are positive whole numbers which cannot be evenly divided by any positive whole number smaller than themselves except 1. The first few primes are 2, 3, 5, 7, 11, 13, 17, and 19. While it is known that there are infinitely many prime numbers, the only way to tell if any particular number is prime is to check it and see. There is presently no practical use for this information other than the amusement of people who play with computers.

The algorithm used to find the first 1000 primes can be described as follows: we give away the first two primes, 2 and 3. We divide each number we are testing by all the primes we have found so far. If any prime evenly divides the test number (ie., there is no remainder), then the test number is not prime and we can go on to the next test number. If, on the other hand, we make it through all the primes found so far without a zero remainder, we have found a new prime and can add it to the list. This is the basic idea, but we can make two immediate simplifications. First, we need only test odd numbers, because the only even prime is 2. All other even numbers can be evenly divided by 2. Second, we do not need to divide by every prime we know, but only those up to the square root of the number we are testing. This is because if there is a number larger than or equal to the square root of the test number that evenly divides it, the result of the division must be a number equal to or smaller than the square root of the test number that also divides it evenly – and we have just ruled all those out! In practice, not all languages support square roots, so we use an alternate method of keeping track of the last prime we need to test. It is not perfect, but it keeps us "close enough".

This is not the fastest possible algorithm (a prime sieve such as that used by Gilbreath, "A High–Level Language Benchmark" BYTE, September 1981, p. 180 would be faster), but it provides a good comparison of the speed of simple calculations in the languages tested.

Here is the algorithm coded in Basic V2.0

```
100 NP = 1000
110 DIM PN(NP)
120 PRINT CHR$(147)
130 PN(1) = 2 : PN(2) = 3
140 TN = 3 : LT = 2 : LP = 2
150 IF TN > PN(LT) * PN(LT) THEN LT = LT + 1
160 TN = TN + 2 : PP = 2
170 R = TN / PN(PP) : IF R = INT(R) THEN 160
180 IF LT > PP THEN PP = PP + 1 : GOTO 170
190 LP = LP + 1 : PN(LP) = TN
200 PRINT TN,
210 IF LP < NP THEN 150
```

The Basic V3.5 and V7.0 tests were run using the same code as V2.0. The Simon's Basic version eliminates the GOTO statement by using the REPEAT..UNTIL and IF..THEN..ELSE constructs. It then appears very similar to the Pascal version of the algorithm.

Here is the algorithm coded in Oxford Pascal:

```
PROGRAM Primes;

CONST totalprimes = 1000;
VAR testnum, primeptr, lasttest, lastprime : integer;
   primenum : array[1..totalprimes] of integer;

BEGIN

 page;
 primenum[1] : = 2; primenum[2] : = 3;
 testnum : = 3; lasttest : = 2; lastprime : = 2;

 REPEAT

  IF testnum > primenum[lasttest] * primenum[lasttest]
    THEN lasttest : = lasttest + 1;
  testnum : = testnum + 2; primeptr : = 2;

  REPEAT

   IF testnum MOD primenum[primeptr] <> 0
    THEN primeptr : = primeptr + 1
    ELSE BEGIN
          testnum : = testnum + 2; primeptr = 2;
         END;

   UNTIL primeptr > lasttest;

   lastprime : = lastprime + 1;
   primenum[lastprime] : = testnum;
   write(testnum:10);

  UNTIL lastprime = totalprimes

END
```

Here is the algorithm coded in Super C

```
#INCLUDE " stdio.c"
#DEFINE totalprimes 1000

MAIN( ){

  INT testnum, primeptr, lasttest, lastprime;
  INT primenum[totalprimes];

  primenum[1] = 2; primenum[2] = 3;
  testnum = 3; lasttest = 2; lastprime = 2;

  DO{

   IF (testnum > primenum[lasttest] * primenum[lasttest]) {
    lasttest + + ;
   }

   testnum + = 2; primeptr = 2;

   DO{

    IF (testnum % primenum[primeptr + + ] = = 0) {
     testnum + = 2; primeptr = 2;
    }

   }WHILE (primeptr <= lasttest)

     primenum[ + + lasttest] = testnum;
     printf(" %d10 " ,testnum);

   }WHILE (lastprime < totalprimes)
  }
```

Here is the algorithm coded in C64 FORTH (immediate mode):

```
3 variable testnum
2 variable primeptr
2 variable lasttest
2 variable lastprime
1000 constant totalprimes
: array <builds 2* 2 + allot does> swap 2* + ;
totalprimes array primenums
: bumpptr 1 primeptr +! ;
: bumpnum 2 testnum +! 2 primeptr ! ;
: notprime testnum @ primeptr @ primenums @ /mod
  drop 0 = ;
: foundprime primeptr @ lasttest @ > ;
: getprime begin notprime if bumpnum else bumpptr
  then foundprime until ;
: storeprime 1 lastprime +! testnum @ lastprime @
  primenums ! ;
: alldone lastprime @ totalprimes = ;
: bumplast 1 lasttest +! ;
: pastlast testnum @ lasttest @ primenums @ dup * > ;
: bumptest pastlast if bumplast then ;
: showprime testnum @ 10 .r ;
: bigloop begin bumptest bumpnum getprime storeprime
  showprime alldone until ;
: setup 3 testnum ! 2 lasttest ! 2 lastprime ! ;
2 1 primenums !
3 2 primenums !
: primes setup bigloop ;
```

The algorithm is simple enough that only very minor changes in coding, if any, were needed to run the test using different versions of the same language. Here are the results (all times in minutes and seconds):

LANGUAGE	VENDOR	print	no print	diff	rel speed	remarks
Commodore LOGO	Commodore	75:03	74:27	0:36	140.72	D,1,5,6
WATCOM Pascal	WATCOM	22:44	22:20	0:24	42.63	C&D,1
Nevada FORTRAN	Commodore	13:25	12:42	0:43	25.16	C&D,2,7
BASIC 7.0	Commodore	11:36	10:34	0:53	21.75	B,1,4,8
BASIC 3.5	Commodore	10:31	--	--	19.72	B,1,4
BASIC 2.0	Commodore	8:29	8:10	0:19	15.90	B,1,4
Simon's BASIC	Commodore	7:07	6:47	0:20	13.34	C,1,5
BASIC 7.0	Commodore	5:28	5:04	0:24	10.25	B,1,4,9
HES FORTH	HES	5:12	--	--	9.75	C,3,5
FORTH 64	Abacus	4:43	3:57	0:46	8.84	D,3,5
KYAN Pascal	KYAN Software	4:13	4:00	0:13	7.91	D,2,4,10
C64 FORTH	Performance	4:06	3:22	0:44	7.69	D,3,4
SuperFORTH64	Parsec	4:00	--	--	7.50	D,3,4
Insta-Speed	Cimmaron	3:55	--	--	7.34	D,2,4,11
Super C	Abacus	3:18	2:58	0:20	6.19	D,2,4
Insta-Speed	Cimmaron	3:15	2:56	0:19	6.09	D,2,4,12
Oxford Pascal	Precision	2:16	2:02	0:14	4.25	D,2,4
Super Pascal	Abacus	1:40	1:33	0:13	3.13	D,2,5
KYAN Pascal	KYAN Software	1:02	0:55	0:07	1.94	D,2,4,13
C Power	Pro-Line	0:49	0:37	0:12	1.53	D,2,4
Assembler	Commodore	0:32	0:26	0:06	1.00	D,4

REMARKS:

B = built-in ROM, C = cartridge, D = disk

(1) Interpreted language.
(2) Compiled language.
(3) Incrementally compiled language.
(4) Stand-alone Basic-runnable programs can be created.
(5) Stand-alone Basic-runnable programs cannot be created.
(6) Estimates only. LOGO ran out of memory at the 306th prime. These estimates are based on time for finding 300 primes: 22:31 with printing, 22:20 without
(7) Nevada FORTRAN runs on the Commodore Z80 cartridge under the CP/M 2.2 operating system. Stand-alone CP/M-runnable COM files can be created.
(8) C128 running at 1MHz.
(9) C128 running at 2MHz.
(10) Compiling to interpreted p-code.
(11) Insta-Speed (also distributed as SpeedWriter by CodeWriter, also known as DTL-Basic) is a Basic 2.0 compiler. Compiling original Basic 2.0 program.
(12) Compiling using speed enhancing options.
(13) Compiling to machine language.

On the whole the results speak for themselves, but we will make a few comments anyway. First, when reading the results bear in mind that comparisons between different implementations of the same language may be more meaningful than comparisons between different languages. The particular algorithm used for this test may not be suited to a given language or not well-coded in that language. If so, then at least within that language the different implementations are still on "common ground" something that might not be as easy to tell with respect to another language.

That the fastest "language" turns out to be assembler comes as no surprise. Hand-coded native machine language is always the fastest executing code on any machine. However, it is not usually the fastest kind of code for a programmer to write. Ideally a high-level language would execute as fast as assembler but be much easier to write in the first place. In practice, although the ease of programming is there, the speed isn't. To use the second fastest language, C Power, a programmer pays a 50% cost in speed using this algorithm (this is not to take anything away from C Power - to pay only 50% is very good, and still 10 times as fast as interpreted Basic 2.0).

At the other end of the spectrum, what accounts for the lethargy of the slowest languages tested - the ones slower than Basic 2.0? We can probably attribute most of the sluggishness of Basic 3.5 and Basic 7.0 (at 1 MHz) to the frequent bank-switching they must do to access the Basic interpreter, text, and variables. WATCOM Pascal is an interactive, interpreted version of a language which is normally compiled, and among other things probably has a lot of overhead calculating at run-time things other versions determine at compile-time. LOGO does so poorly because we've assigned it a task which is a long way from the language's strengths: list processing and powerful, simple graphics. LOGO is like a simplified LISP (LISt Processing), the premier language of artificial intelligence research, in the same way Basic is like a simplified FORTRAN. The design needed by a language with good list processing primitives - most likely each list (indeed, each LOGO variable) is a binary tree whose nodes contain literal character strings - virtually guarantees both the memory and speed problems we encountered. In the case of Nevada FORTRAN, problems of bank-switching, interpretation, and inappropriate language design do not apply. It is possible that the lack of speed results from the hardware - the Z80 is less efficient than the 6510 at some operations, and on the C64 it runs at a slower clock speed than it is capable of. On the other hand, perhaps the FORTRAN compiler simply produces inefficient code.

A quick and dirty survey like this may begin to answer the question, "Which language is fastest?". The answers to a number of other relevant questions that might be considered are not provided here, however. There is no description of how easy or difficult a particular implementation is to work with, for example. For languages with a separate compile step there is no indication of how quickly the compiler does its job or the size of the finished program file it produces. Many of the languages have official or unofficial "standards" that describe what should be included and how things should be done. Our survey only says that the implementations of each language we tried were complete enough that we could run our test; it does not indicate where a particular implementation falls short of the standard or has a non-standard extension of some kind. Unique or unusually helpful features of some implementations have not been mentioned. Questions of cost, availability, vendor support, and licensing arrangements (in the case of compilers capable of producing stand-alone Basic-runnable program files) have not been addressed.

Nevertheless, we think our results are interesting and would welcome additions to our list!

CP/M Block Allocation Calculator

Miklos Garamszeghy
Toronto, Ontario

On the C–128, CP/M single sided disks are divided up into 170 logical blocks, called allocation units, of 1 K bytes each. The allocation unit is the minimum amount of disk space that CP/M will assign to a file or add to a file as it grows. Each allocation unit consists of four physical disk sectors. Double sided CP/M disks use 2 K byte allocation units, consisting of eight disk sectors. Unfortunately, the pattern of sectors used for each allocation unit is rather complex and is quite different from the normal filling order of COMMODORE DOS blocks. CPM BLOCK is a short BASIC 7.0 program that will calculate and print out a list of the track and sector numbers corresponding to each of the 170 allocation units for either single or double sided disks.

The list can be used in conjunction with a disk sector editor, such as "DISPLAY T&S" on the 1541 demo disk, to trace the contents of a CP/M file on the disk. The CP/M directory is located in allocation units 0 and 1. Data starts at allocation unit 2 and normally filled in consecutive order. Files on disk that have been written to and scratched a number of times may be scattered widely over the disk. The allocation unit numbers used in the print out correspond to the numbers found in the allocation map (bytes 16 to 31) of each directory entry. For double sided disks, CP/M fills side 0 first then side 1.

All single sided CP/M disks, C–128 included, are divided up into 1k byte logical areas called blocks or allocation units (AUs). The allocation unit is the smallest space on the disk that a file can occupy. For example, even if a file contained only one byte, the other 1023 bytes in its allocation unit cannot be used by another file. The C–128 single sided disk contains 170 allocation units, numbered 0 to 169. AUs 0 and 1 contain the directory, while the rest are used for data storage. Each AU is logically subdivided into 8 "records" of 128 bytes each. The record is the smallest addressable unit for finding or storing data on a disk within a CP/M file. As files grow, they contain more records and consequently more blocks are allocated from the list of empty blocks.

Since the C–128 sector size is 256 bytes, each CP/M AU is comprised of four physical sectors on the disk. The actual structure of the C–128 CP/M disk is the same as a standard C–128 disk in terms of sector size, number of sectors per track and number of tracks per disk. The order in which the sectors are filled, however, is quite different. The sector skew rate is 5. This is easiest to visualize if you think of each track as a dartboard with the segments numbered in consecutive order from 0 up to the maximum number of sectors on that track. The sectors are filled starting at 0 and jumping 5 each time to the next. That is 0, 5, 10, 15, etc. When you complete the circle once, you should have gone past the 0. For 21 sectors per track you will end up at sector 4. The cycle then repeats: 4, 9, 14, 19, 3, 8, 13, 18, etc. until all the sectors on the track have been used and it jumps to sector 0 of the next track. For any given track, this can be expressed by the simple cyclical relationship:

next sector# = last sector# + 5: IF next sector#>maximum# on track THEN next sector# = last sector# + 5 – number of sectors on track

From this relationship, you can see that the actual filling order of the sectors depends on the number of sectors on the track. Thus, there are four distinct filling orders used on the disk: one for each of the four areas on the disk with different numbers of sectors per track. Track 1, sectors 0 and 5 as well as track 18, sector 0 are reserved for special system functions and are not included in the fill table. For reasons unknown to humans, Commodore decided to structure the CP/M disk as 640 logical tracks of one sector each. The numbering of the logical sectors corresponds to the fill table. Double sided C–128 CP/M disks have an allocation unit size of 2 k bytes or 8 physical sectors or 16 records. Side 0 is filled first, then side 1 in basically the same order. Track 36 sectors 0 and 5 and track 53, sector 0 (corresponding to the unused sectors on side 0) are not used on side 1.

Basic Source Listing

```
JC  100 rem save "0:block calc",8
EO  110 print cs$"** cpm block calculator **"
PN  120 print "    by m. garamszeghy"
FN  130 print "    ver 2   86-04-01": print
IC  140 cl$=chr$(157): cs$=chr$(147)
OP  150 input "<s>ingle or <d>ouble sided";sd$
        : sd=1: if sd$="d" then sd=2
DE  160 input "<s>creen or <p>rinter";ot$
KM  170 if ot$="p" then open 4,4: cmd4: else print cs$;
DL  180 t=1: s=10: tc=2: sm=20
OF  190 print " 1571 cp/m disk block map "sd" sided"
FH  200 print "block# ",,, "track:sector"
NL  210 print "dec (hex)";: sk=1
KB  220 for bc=0 to 169
KF  230 print: print bc;cl$"("right$(hex$(bc),2)")",
FO  240 for sc=1 to 4*sd
JJ  250 print t;cl$":"s,: s=s+5: tc=tc+1
        : if s>sm then s=s-sm-1
KM  260 if tc>sm then t=t+1: s=0: tc=0
        : if t=18 or t=53 then s=5: tc=1
DP  270 if t>17 then sm=18: if t>24 then sm=17
DN  280 if t>30 then sm=16: if t>35 then sm=20
MC  290 if t=36 and sk=1 then s=10: tc=2: sk=0
FC  300 if t>52 then sm=18: if t>59 then sm=17
        : if t>65 then sm=16
GC  310 next sc,bc: if ot$="p" then print#4: close4
```

Compatibility And Operability Of The C128 CP/M+ Operating System

Ralph A. Morrill
Mercer Island, WA

© Copyright March 1986

In being so busy selling "features", CBM had dramatically understated the full CAPABILITIES of this Mean Machine!

Prolog– To CBM

In July, 1985, we purchased our first C–128 computer, upgrading after three years of use of a C–64 for our routine business operations. We are a small R&D consulting firm dedicated to the propagation of the considerable talents of our principal to a distinguished list of clients, which includes top Fortune 500 firms. Most of our services are conducted on large mainframes and minicomputers, but we have gained considerable respect for the capabilities of the CBM microcomputers through practical business use of the C–64.

Our first C–128 (July '85) was a beta unit, distinguished by a two column grey right border on the graphics screen, and improper vertical striping of all VIC output in odd color assortments. A replacement was not available until October, during which time, denied full use of the 64 and 128 modes, we began evaluating the CP/M mode. Ours was one of the first of many letters to CBM pointing out the numerous "bugs" and deficiencies in the first two issues of the CP/M+ operating system. Long telephone conversations with John Fahey, CBM customer relations, about our early findings resulted in a formal invitation in December to beta–test the 6 December '85 revisions and CBM future upgrades to CP/M+.

Two preliminary "Letter Reports" were submitted on a timely basis, allowing CBM sufficient time to respond to our recommendations. This is our final report to CBM of the results of our tests and evaluations. It has not been paid for by CBM, nor will it be. It is therefore PROPRIETARY COPYRIGHTED information and the property of RAMA Corporation, intended for public release to the media catering to Commodore computer users. It was submitted to CBM in March, 1986, for comment, as a courtesy, before formal publication.

Introduction

Like Christmas, there is always a sense of tense excitement when unwrapping your new Computer, be it the first, third or nth. It was July '85 and I had anticipated this moment since last February when the first public information releases had been made on the new Commodore C–128 microcomputer. The decision had been made then to upgrade our business computer system with the new C–128. I had read everything available, even written a brief article about this "New Three–In–One" for a local Commodore user group.

FINALLY! It had arrived and I eagerly got it out of the boxes and quickly connected it to the existing peripherals (a compatibility unusual in itself). The old C–64 was laid aside, retired from its three years of daily fault–free service as our busy business computer. I connect the 40/80 column screen switching box (made a month earlier), then the modem, high–res composite color monitor, printer, and two, new 1571 "smart" disk drives. Now, let's see what this beauty will do!

I doff my hat to the sage who once said "anticipation is nine–tenths of the pleasure. . .", however, in this case, he was wrong. The C–128 was more than Commodore had said it would be (and I had read every golden word). In being so busy selling "features", CBM had dramatically understated the full CAPABILITIES of this Mean Machine! It was perfect for our business use, and a great deal more.

The uneasiness caused by the opening screen – which states that BASIC 7.0 was Copyrighted in 1977 by MicroSoft, and updated by CBM in 1985 – soon disappeared as I began to try some of the BASIC features provided. Help, Tron, Sound, Graphics, Sprites and the new BASIC 7.0 commands made programming effortless – all greatly enhanced by the 40/80 column screens, extended Control and ESC codes. I BOOTed the CP/M+, which I hadn't played with for years (V 1.4 as I recall), and discovered a near CP/M–86 capability imbedded in the calls that was approaching a REAL "Operating System", that CP/M never was before.

I had purchased an early beta–test unit, distinguished by a small brown dot under the "M" in the word COMPUTER on the name tag, and a grey stripe on the right side of the graphics

screen. This one had a ROM fault that caused random character errors and also had a bad VIC chip. Our software worked all right for business use, but the graphics were vertical striped in an odd assortment of colors. This didn't bother me much, having accepted it as a high probability for one of the first units sold, but a replacement would not be available until October.

Discovering CP/M+

Denied the full use of graphics in the 64 and 128 modes of operation until a replacement unit was received, I concentrated on the fully independent CP/M mode, learning all there was to know about the CBM upgrades to CP/M 3.0, and ordering the "DRI Special Offer". The latter is one of the best buys in the country, and probably the ONLY "support" we will ever receive from Digital Research Incorporated (DRI) for their operating system. The "Two Utility Disks" in this DRI offer are more than just that; they contain the assembly level ASM/LIB files, macros and object codes – plus the necessary MAC, RMAC, SID, ED, etc., utilities which are essential if any programming is to be done, or public domain software installed on the system. The two inch thick, three–in–one manual(s) that comes with these disks, also an essential, is cryptic to a fault; but I would hate to do without it. It sort of explains the functions (NOT the functional use) of the utilities and reveals the content and details of the BIOS, BDOS, CCP and other incorporated subroutines of the operating system. A fourth manual, "Programmer's Utilities Guide for the CP/M Family of Operating Systems" referred to frequently in the above manual(s), is not available from DRI's publisher for separate purchase. Rather than pay an outlandish $75 for repeated purchase of the manual(s) materials, I recommend the Osborne/McGraw–Hill CP/M Users Guide, Third Edition. CBM's Manual and Programmers Guide should have eliminated this serious problem, but did not.

The first thing to do when starting any new system is read the File Headers and DOC files. The HELP file on the first distribution disks serves as a reference until the "DRI Special Offer" arrives – although it is just as cryptic as the DRI manual(s). There is nothing you can do with all these files until you obtain some applications software or the essential utilities for programing contained in the DRI utilities disks. You're stuck, except to study what you have on the original distribution disks, and try them out against each other. Furthermore, the first two CP/M+ issues by CBM did not permit a required DEVICE assignment to a modem (RS232 Port) – and thus NO telecommunication and down–loading of applications programs is possible. Remember, the big sales pitch by CBM for the CP/M+ operating system was the "Free Software" available in the public domain – now inaccessible.

Discovering Problems In CP/M+

I tried the simple exercises of the CP/M+ operating system and utilities and fortunately, all the utilities DID work properly (I have a few choice words later to describe my feelings about the archaic SID and ED file editors). The first, most obvious problem discovered was that the printer would not properly copy (echo) the screen, no matter what I did to the interface configuration switches. This was later revealed to be a problem within the interface (it works fine in 64 and 128 modes) that was not designed to handle the eight–bit ASCII or PET–ASCII codes (nulling the high–bit), and was "coloring" the printing with odd "soft–returns" mid–line and mid–word. I also had some random graphics and font switching codes being sent to the printer at carriage–returns. Asking others about their printer problems revealed that only the Cardco Plus and Plus–G interfaces did not have all these CP/M+ problems, although upper–case characters were still being printed in italics. I had to get rid of the PET–ASCII and print with the interface in transparent mode.

Further testing revealed that, although the intent of CBM was to default (Boot) the system in an ADM–31 terminal configuration, the BIOS KEYCODEs were wrong, resulting in a keyboard that, at best, was an ADM–3A terminal configuration plus some odd ESC codes unique to the C–128. The Console Command Processor (CCP) was also deficient, and trying to install and run commercial applications software was, in many cases, a disaster. In some, the opening screen and menus were in all graphics characters (the alternate key–set produced by ESC–G–1) or resulted in a resounding CRASH because they wrote buffers into high memory overwriting the CCP. Furthermore, random character errors kept showing up on the screen and printer outputs, along with double characters. From this some of the most annoying and unpredictable crashes resulted. At that time I was compelled to advised CBM of my consternation and frustrations. My letter was not complimentary; I learned later that it had been read to the Board of Directors and circulated to CBM management. The reply (I actually got one) was from the CBM legal council denying all responsibility. (Naturally! That's what he is paid for.)

When the "DRI Special Offer" finally arrived six weeks later, the READ–ME file was not the least bit illuminating, but the MAKEROM.DOC and MAKESYS.DOC files did guide me through the exercises of assembling and linking the object codes that fill these disks. DO NOT try to make sense of the file names and headers with the DRI manual(s) – there is NO CORRELATION between these, (this information should have been in the READ–ME file, according to the DRI Manual). This lack of correlation was the source of enormous frustration to me and my innate compulsion to MAKE IT WORK! The second curiosity was that although the MAKESYS exercise did, in fact, assemble and install a new (and supposedly improved) CP/M+ system, ALL the faults discovered in the first issue prevailed – including the denial of use of a modem for down–loads. In addition, the MAKEROM exercise dead–ended with a set of BIOS "CXROM. . .." files of absolutely no utility or reason, since they had already been included in the MAKESYS files under other/different object code file names. DRI refused to answer

my questions, sending me back to CBM's hot-line mummies. The author, finally reached directly, admitted to me that this mess was intentional, not an accident or oversight, in the process of developing these files. It does provide us with a fairly complete set of the CP/M+ object codes, and we may all be grateful to him for these in any future upgrades of our CP/M+ operating system.

Invited To Beta-Test CP/M+ Revisions

I tried using SID and ED to find and make corrections to the CX. . ..ASM files that governed the functions that were faulty in the BIOS/BDOS/CCP codes, and decided this was an exercise in futility without very extensive study. Did you ever try to edit someone else's code, on a limited display screen, without seeing the cursor position and without having full screen edit capability; it is AWFUL!! Furthermore, some of the most dramatic crashes imaginable resulted from these efforts, resoundingly slapping my sensitivities as to my own capability as a programmer (now seriously in doubt). Once again, I let CBM know about my concern and frustrations with this situation, and was invited to help constructively solve them as a "Beta-Tester" for the third issue of CP/M+ which was alleged to solve all these problems. It would accommodate the modem, RAM disk and other DEVICE assignments, and much, much more. Having already developed a small affection for this "Mean Machine" - I agreed to beta-test the CP/M+ revisions.

This latest CBM revision of CP/M+ (6 Dec. '85) for the C-128 was to be a fully corrected version, removing of all the faults I had cited above. It would now allow modem (RS232 port) operation in CP/M for down-loading software that would make the beta-tests a significant exercise, and covering a wide range of utilities and applications software. I immediately sent some 38 letters to all the top CP/M software suppliers (very few of whom responded) for test samples. I also promptly down-loaded (in 128 mode) and converted the public domain release of the 4 Dec. CP/M+ BIOS, and went to work in getting a first-class terminal (MODEM7) program up and operating. Finding a properly overlayed IMP244.LBR (which I highly recommend, by the way) was a bit of a struggle, but it had just been made available on CompuServe. My down-loading began, and over the next few weeks - while waiting for CBM to send me the Beta-Test issue of the CP/M+ BIOS - I accumulated an outstanding set of utilities and software.

The First CP/M+ Revision

The disk that arrived from CBM in January was folded into a U and stuffed into our postal box. With heart-sick feelings about the condition of the disk (and a bitter word or two to the Postmaster), I hurried back to the office and was pleased to find I could gently unfold it and use the disk in the 1571 drive - copying everything promptly to a new disk, just in case.

Along with a new CPM+.SYS file was a SCREEN40.COM file that I investigated immediately. This little ditty solved several serious problems. It turned off the 40-column screen and completely eliminated the random character errors experienced above, particularly in 1200 baud modem transfers. The source was thus revealed for at least one of the faults in the CCP/BIOS - although SCREEN40 is no more than a Band-Aid for more serious problems that were still internal to the CP/M+ operating system. My down-loads from the BBSs would now be clean of random errors that plagued our first test software. About the same time, the C-128 "Configurator" CONF.COM and its menued HLP file became available on the public domain. With the proper settings of the O.S. using CONF.COM, and the printer interface set transparent to eight-bit ASCII, my Star Delta 10 printer properly receives and prints the direct ASCII codes without interference from the interface. Calling CONF PRT1=ASCII turns off the PET-ASCII conversion (that has been fouled up in the CP/M+ codes) needed for CBM printers. Calling CONF 40COL=OFF turns off the 40 column screen (like SCREEN40), goes to two megahertz, and gets rid of most of the random character errors at 1200 baud, and double keying problems. There is a full menu of other configuration control and set-up calls in CONF.COM, as well (see below) that provides the needed configuration control of the system - including screen colors. The controls provided by CONF.COM are REQUIRED for the successful use of the 6 December issue of CP/M+. They are only a Band-Aid "fix", however, to the many serious problems that are still imbedded in the CP/M+ source codes.

With the improvement of the printer operation (by setting the PRT1=ASCII in CONF.COM), I realized my first real success with echoing the screen and running several NLQ font programs that bit-map output to the dot-matrix printer. I use the "CBM emulate mode" of the interface only in 64 and 128 modes now, with the software designed for CBM printers. With the beta-test CP/M+ you must still configure commercial software for the ADM-3A terminal configuration, and thus are denied use of the full keyboard features and the much faster screens of ADM-31 terminal operation. Since this is not a hardware problem, but a fault of the CP/M+ codes, we should expect CBM to properly correct this problem. Several of the commercial and public domain software packages evaluated require that the CCP be overlayed or made a PRL (program relocatable) file to work with their resident commands. As the CCP MUST be reprogrammed to correct remaining errors in its codes, WE NEED an assembly level copy of the CCP (i.e. CCP.ASM or CPR.ASM) to successfully run this excellent new software. Several Z80 disassemblers I have tried crash the CP/M+ O.S. - although nearly all CP/M 2.2 or 3.0 commercial software and utilities run great on the C-128 (in ADM-3A terminal mode with the Band-Aids affixed to the O.S.).

Not one piece of Osborne software from the First Osborne Group's (FOG) public domain library that I have acquired will run on the C-128. Yet nearly all CP/M-80 software from the

SIG–M Library, and so called "vanilla" programs (i.e. without special machine unique calls imbedded) down–loaded have properly run. (I do not recommend that the C–128 CP/M+ be reconfigured to accommodate Osborne software – even the Osborne Executive's CP/M 3.0) All Osborne, QX–10 and Kay-Pro disk configurations DO load and copy properly (even between disk formats) with the 1571 disk drives. As a matter of fact, the 1571 is remarkably fast and fault free in the CP/M mode (although we cannot say the same for the 128 mode where the DOS fails periodically). Although our C–128 unit does have the infamous "shifted–Q" problem, our C–64 software is 100% compatible. Even the fully loaded Flight Simulator II will run without having to disconnect the printer interface and cassette ports (thanks to CBM for cleaning this mess up).

Software Reports: Successes

The following abstracts are briefs of the software I have tested that does operate successfully on the C–128 (under the certain conditions stated above and with the band–aids) with the 6 Dec. beta–test issue of CP/M+. We anticipate that CBM will make all necessary changes to the BIOS and CCP, and correct the remaining faults discovered in the codes, for the final release of the upgraded CP/M+ operating system scheduled for May of this year (now 5 months late). I will not belabor this point further here. Only positive results are presented below, but rest assured, there were many failures.

Modem Programs – Public Domain

MEX114.LBR – Contains an excellent terminal program (needs jump table edits to work on the C–128). A full set of utilities and large set of overlays for various systems and modems are contained on several disks. The C–128 overlay has been installed on one version available on local BBSs, but only for the CBM 1670/Hayes modems. It is loaded with set–up controls and toggles, for those who like this feature, and will batch transfer in several protocols that include CompuServe's. It is very well documented and available on SIG–M Library Vol #218, 219, 220 and 241.

IMP244.LBR – Another excellent terminal program with all the features of MEX – but most are automatic and transparent to the user. It also will batch transfer 128, 256 and 1K Byte Blocks and has KMD or XMODEM (Christensen) protocols. I much prefer this one (it's fool proof), and have recommended that CBM license it and adapt it to all CBM modems. There is an C–128 overlay (I2C8–1.OVL for the 1670) and a converted version (IMP–128.COM) available on most local BBSs. The needed utilities are included only in the LBR file, so get both. It is very well documented, but the overlays must be ordered from the author.

Utilities From The Public Domain

CONF.COM/HLP – The C–128 configurator. These files MUST be on the CP/M+ update disk. It was released to the public domain on 21 Jan. 86 by the author, Von Ertwine, and has been a life–saver. The command and help file support the set–up of system:

- 40COL : toggles on/off the 40 column screen (and eliminates random character errors).
- BAUD : sets the baud rate for the RS232 Port.
- BACK/BORD/CURSOR : set colors on the screen.
- DATE : sets system date and time.
- DRV : assigns disk drive device numbers to A:,B:, etc.
- DUMP : dumps 16 bytes of memory from ROM in hex from a selected address.
- FEEL : sets key scan frequency.
- HELP : types the help menu and instructions
- MAP : maps the keyboard assignment by KEYFIG.
- PARITY : sets byte character (ie 8/N/1, etc.)
- POKE : pokes a hex code into memory at selected location.
- PRT(1 or 2) : sets printer output to ASCII or PET–ASCII and selects character set.
- REPEAT : toggles repeat keys on or off.
- VOL : sets audible key–click volume level.

NULU151.LBR – Menued and well documented LIB file utility. Makes and un–makes libraries with squeezed files. Has features for reading, extracting, squeezing/un–squeezing, making and erasing files. You will want to find the COM and DOC files separately to get started (or get USQ.COM and DELBR.COM files to open this LBR).

NUSWEEP107.LBR – Library file utility with some features not found in NULU. Well documented and menued for ease of operation. Actually, both are needed and compliment each other. Both will allow indexing of files and in one "sweep", extract only those files selected – un–squeezing as it goes.

USQ/DELBR/BISHOW.COM – A nice set of individual LIB file utilities contained on all SIG–M Disks. Unsqueeze/Extract (ie de–library)/Read files in squeezed or un–squeezed format, respectively. BISHOW (or QSHOW) out performs the TYPE-.COM utility on the CP/M+ distribution disk.

CPM3UTIL.LBR – Excellent set of CP/M+ utilities found in SIG–M library volume 234. Contains the following:

- DISK3.COM : full directory listing and available space.
- EDIT3.COM : Text–editor enhanced over ED.
- IMAGE.COM : track by track disk copier, any disk format.
- PASSWORD.COM : to protect the CP/M+ System in PROFILE.
- UNLOAD,COM : converts binary file to Intel hex file.

- VERIFY3.COM : checks all disk sectors – any format.
- DISKED3.COM : menued sector editor for direct read/write.
- WRTSYS.COM : writes boot to system tracks of disks.

UNERA+.LBR – Poorly documented, it restores erased files.

SUPERZAP.LBR – Full–screen disk editor, with menus.

CPM3–CAT.LBR – Disk catalogue maker.

BRADFORD.LBR – NLQ printer software like fancy font. Few fonts, not documented, "freeware", but worth it.

Commercial Software Tests

WORDSTAR 3.3 – Latest version of the word processor standard. Do not bother with earlier versions. Good books are available to cover documentation needs.

WORD FINDER – Thesaurus for use with WordStar. Excellent to check for spelling as you type the document.

SUPERWRITER – Fine word processor with spell checker and dictionary. Works with SUPERCALC2 to print full reports.

SUPERCALC2 – Great spreadsheet software that has Data File Format Converter included for CP/M–86/80 formats. Works with SUPERWRITER to produce full reports.

FANCY FONT – Excellent NLQ printer software in many type styles and almost any ASCII file format. A very large library of fonts are available, in sizes from 4–72 point. Very well documented, bit–mapped and S L O W !!

TWIST & SHOUT – Sideways and banner printing utility for dot–matrix printers. Few fonts compared with print shop.

TURBO–PASCAL – A must for any CP/M+ system. Compiler, editor and other utilities. Makes the public domain T–Pascal software useable.

C/80 3.1 & MATHPAK – C compiler and utilities for C language programming. Well documented and recommended.

GAMES – We tried ZORK II, Adventure 80, SNAKE, ELIZA and some of the other CP/M text games, with some limited interest. Not as exciting as the C–64/128 arcade games, to say the least.

COMPUTER CHEF – The main software of a set of four recipe disks and meal planners. As an amateur chef I liked it, particularly the Chocolate Bytes Library. Auto–sizes recipes to the number of guests.

ZCPR3.LBR – I'm still in the process of evaluating this MASSIVE set of files (seven disks full) to replace the CP/M CCP and expand the limited capability of the operating system, Dramatically! It was written for CP/M 2.2, and will take a great deal of time to install. I will report on this separately, later. I HIGHLY recommend that CBM take a long and serious look at this outstanding option.

Wrap–Up Of This Report – To CBM

We have taken the process of beta–testing of the CP/M+ upgrades to their logical conclusion consisting of the preceding series of recommendations to CBM for needed correction of several problems. The software tests are by no means completed, but will continue and be published as appropriate for unique software packages made available.

Our recommendation that CBM fully investigate ZCPR3 implementation is taken seriously by Exchelon in California; it has promise of making a Magnificent Monster out of this "Mean Machine". The original CCP is a mess inherited from an archaic CP/M concept. With its replacement, ZCPR3, the capability of the C–128 will be greatly expanded to include; shells, aliases, I/O redirection, flow control, named directories, search paths, custom menus, multi–command lines, many resident commands, and much more! With it we will realize the UNIX–like operating system capability from this microcomputer that exceeds any other available on the market today in this price range. A "Much Higher Intelligence – at a Much Lower Price" – indeed!

On the hardware front, we do highly recommend that an expansion package be made available, from CBM, to add two more banks of 64K bytes of RAM/ROM in the "empty socket". The upgraded CP/M+ BIOS should accommodate this capability. Your promise of a GEM capability will thus be realized, and the full utility of the C–128 for us, the business community of users, will be greatly enhanced.

We have enjoyed performing this extensive research and evaluation exercise for CBM, and their loyal following of Commodore computer users. We have recognized the enormous potential of the C–128, better than most; perhaps better than CBM at this time.

Assembling Assemblers

Chris Miller
Kitchener, Ontario

Chris Miller is the author of the Buddy System–128 and Buddy System–64 Assemblers from Pro–Line.

Writing an assembler is like no other programming activity. It has its own peculiar set of rewards and challenges. I would like very much to pass on some observations, experiences and solutions to readers who have decided to do their own from scratch or to build on the Symass assembler as published in Transactor.

It's Alive

Keep archives. Don't be in too big of a hurry to update everything. Unlike almost any other type of program, an assembler lives; it creates itself. Accidently killing it can leave you in a most peculiar bind.

Suppose you've got several up–to–date backups of version 12.3 when you discover a tiny flaw in your error checking. No problem; the fix is obvious and after making it you quickly re–assemble new version 12.4. In your excitement over having finally taken out the last bug (again) you replace all your old, slightly imperfect 12.3 source and code with your new, at–last–perfect 12.4 stuff. Six minutes later you think of a great new command to add. It's an easy update: a flag, a pointer, a little code. . . but when you go to assemble it every single line of source gives rise to an INVALID MODE OF OPERATION! error message. You see, you forgot to include a crucial RTS when you repaired the last version.

Panic sets in (as well it should). There are so many custom macros and features in your source by now that no other assembler can touch it. If you're lucky you may find an old version 6.1 tucked away that you forgot to update way–back–when that, although far from perfect, can still do the job. Otherwise, you may have to spend a couple of days bringing your source down to the level of whatever assembler you started with and working your way up again.

It is also possible to have only wounded version 12.4 so that, even though it seems to work, inaccurate code for version 12.5 is output and it isn't until you go to assemble version 12.6 (or even much later on) that everything actually blows up in your innocent but startled face. So keep a few of the older generations around awhile, eh.

BASIC's Terrible Tokenizer

If you want to be able to write your source on the Basic editor, sooner or later you are going to have to do something about tokenization. It is not enough to simply change the appearance of mnemonics like EOR, AND and ORA and to tell people to use $DEE+1 instead of $DEF. Consider the following perfectly legal source lines:

```
20 remove =*
30 testdata ora #8
40 newdata jmp newline
```

In line 20 the "=" and "*" would hold their ASCII instead of their tokenized values so that your assembler would not recognize them. The ORA would not be tokenized by the editor in line 30 so that your mnemonic lookup routine would never find it. Even though NEWLINE was defined, line 40 would give rise to an UNDEFINED SYMBOL error. If you are not going to un–tokenize source before assembling then you may not use "REM" or "DATA" in any symbol name.

Personally, I have tried everything to get around BASIC's idiosyncratic tokenizer, including trying to live with the above restrictions, placing all source within quotes and monkeying with the editor itself, none of which proved satisfactory.

The thing to do is really not all that difficult: every line of source must be un–crunched into a buffer by the assembler before parsing. There is a Basic routine to do this and it is actually very fast. The time lost in un–crunching is also compensated for by the fact that you can now index source lines with both the .X and .Y registers and that absolute addressing modes are faster

than indirect. This also removes all restrictions imposed on your source by tokenization, makes your assembler easily compatible with ASCII formats, and opens the door to disk based assembling and easy display handling.

Look–Up Look–Outs

The Symbol Table.

The most time–consuming aspect of any assembly (excluding I/O) usually involves building and accessing the symbol table. A common structure involves allotting a fixed number of bytes for each symbol (usually eight) plus two for the value and stacking entries one–on–the–other (usually in a downward direction) in memory. The advantage to this system is simplicity but there are two major drawbacks:

1. It can be very slow, especially once you begin chaining source files and symbol tables become very large. If you do re–definition checking (almost a must) or phase alignment checking on pass two, a large, sequentially organized symbol table will slow things down even more.

2. Symbol names must be restricted in size (I find eight to be not quite enough) or a good deal of space must be wasted by constantly providing for longer symbols. Consider variable length entries: let the first byte of each entry represent its size. You will be able to support unique symbols of any length and conserve memory too. Of course, speed will not improve.

To really speed up the old assembler, try organizing the symbol table like a binary tree. Each entry will, in addition to the name and value, require space for two pointers (four bytes). Following is some pseudo code for placing a symbol in the table.

```
put'symbol
initialize entry'ptr (to root of tree)

compare'next
  compare new symbol with entry

if same then re–definition error

if before (in alphabet)

then

  if left'pointer = nil

    then (not in table)
      left'ptr = end'of'table'ptr
      goto tack'on'entry

    else (branch left)
      entry'ptr = left'ptr
      go to compare'next

else (comes after in alphabet)

  if right'ptr = nil

    then (not in table)
      right'ptr = end'of'table'ptr
      goto tack'on'entry

    else (branch right)
      entry'ptr = right'ptr
      go to compare'next

tack'on'entry
  put new entry at end of table
  new left/right ptrs = nil
  update end'of'table'ptr
  return
```

There are probably quite a lot of ways of coding the above idea in assembler; far be it from me to deprive you of the pleasure. It will usually be necessary to examine only a small portion of the tabled symbols to find one or determine that it is undefined. For example, if you were trying to find ZEEK in the table and the first entry you compared it to was OTIS, then you would never have to check any entries which came before OTIS in the alphabet after that. Speed of access makes a binary symbol table structure worth a few extra bytes of overhead.

Command Look–Ups

As with symbols, there are many ways of organizing the instruction data. A sequential list of fixed length entries is again the simplest, but also the slowest and the greatest waster of memory. It is slow because any three–character symbol will require a complete search of all the mnemonics from ADC to TYA before you can be sure that you do not have a command. Even if you try to keep the most popular instructions towards the top of the list, it will often be necessary to scan through 50 or 100 of them until you find the one you're after. It's a waste of space to follow each mnemonic, especially the inherent ones, with a whole slew of filler bytes representing invalid modes of operation surrounding usable op–code byte(s).

Pointers to Lists

Consider following the mnemonics in the look–up table with a couple of address pointers: one pointing to a list of valid modes,

the other pointing to a corresponding list of op–codes. Attach a (symbolic) value 1–12 to each mode. Suppose the assembler runs across the statement LDX (PTR),Y which we all know can't be done. Parsing the operand portion determines that indirect indexed is being used. The mode'pointer list for LDX, however, does not contain the value (e.g. 4) for this mode. You know this as soon as you skip past from zero page (e.g. 2) mode to absolute (e.g. 6) or the end–of–list flag (e.g. $ff) is reached.

All of the inherent commands will point to the same two–byte list of valid modes (e.g. 1, $ff). Many of the other commands will share the same list of valid addressing modes as well. Of course, the opcode'ptr for each mnemonic will point to a unique list of op–codes, however, no filler bytes will be required because this list will correspond exactly to the list of valid modes. If the mode value you were looking for was third in the mode list, then the op–code you should use will be third in the op–code list. Let's do something with some of the memory you just saved.

Pointers to Pointers

Instead of one huge list, try breaking the mnemonics down into a bunch of small lists based on the first letter of the command. When you go to look up JMP, the value for "J" can quickly be converted to 9 (e.g. LDA #"J":SEC:SBC #"A":AND #15). Indexing a list of low byte and another of high byte pointer values by this should provide you with a pointer to the list of mnemonics which begin with the letter "J" (two in this case). You'll never have to look through more than half dozen or so.

There other ways of massaging the 6510 mnemonics to produce even more, smaller tables for even faster hash accessing. I challenge anyone to come up with an algorithm that will generate a unique, one byte value for every standard 6510 mnemonic.

Output To Disk

Sooner or later it will become necessary to direct your output to disk. For one thing it is not always possible to assemble your code to the memory that it is destined for. An assembler attempting to assemble itself to memory will overwrite code that is being executed; risky business.

The easiest way to send everything to disk is to open the file from Basic before the assembler takes over via 5 OPEN 5,8,5,"0:NEATCODE,P,W" for instance, and then just use the Kernal $FFD2 (Chrout) from within the assembler. Be sure to also send out the value of the program counter before the first byte is sent if you want to LOAD"NEATCODE",8,1. It is a small matter indeed to create a command to set a flag which will tell the assembler what to do with the object code on pass two.

Much better of course would be to have your command actually open a file itself on pass two and if necessary close the previous one. This way more than one object file can be created at once, and if there is an error on the first pass, you can stop before the file is created. Reading the error channel and always closing files is also polite.

Hidden RAM

Bet if you take a hard look at the Basic ROM routines you're using, you'll find that you can re–create and even improve on them yourself. Then you can put your assembler under Basic Rom from $A000–$BFFF. You'll need to turn bit–0 at address 1 off and on from a safe place before starting and after finishing an assembly. You could also put a C–64 FARJSR relay routine to call Basic and Kernal ROM from their underlying RAM. Maximize the amount of RAM free for source, symbol table and testing.

You're Hooked

Once you've come this far, there is no turning back. You're bitten. What about a command to cause a specified source file to be loaded into memory before continuing from the top. After all, you can't expect to keep all of your source in memory at once forever (even in the 128). Or what about commands to Load and Save symbol tables for immediate use by the assembler as an alternative to the above sort of chaining. What about conditional assembly? What about MACROS?

Writing an assembler can be a very useful, educational and even profitable experience. I wish you success.

Structured Programming on the Commodore 64

Frank DiGioia
Stone Mountain, Georgia

The power of Pascal with the ease of BASIC

This article presents a BASIC language extension which will allow you to write structured, Pascal–like programs in BASIC 2.0. This exciting capability will make it possible for you to use your C64 to write programs for school, business and other situations where normal, unstructured BASIC is traditionally considered inappropriate.

There is a lot of talk these days about structured programming. You won't find a computer science department in any school that doesn't emphasize the importance of such programming techniques. Structured programming isn't just limited to universities, however. Most large corporations require that all of their Information Systems employees use structured design and programming techniques in all of their work.

Well, with all the emphasis being placed on structured programming, it makes sense that you might want to practice some of these techniques on your own computer at home. Unfortunately, it is very difficult, if not downright impossible, to implement structured programs on the Commodore 64 due to the limitations of BASIC 2.0. In order to allow you to practice structured programming on your C64, I've designed a BASIC extension program which will implement some commonly used structured programming commands on your 64. Before going into a description of these new commands, however, I should describe exactly what I'm referring to when I speak of structured programming.

TOP DOWN DESIGN

The two most important concepts in structured programming are TOP DOWN DESIGN and MODULAR PROGRAMMING. Very briefly, top down design refers to the idea of taking a task and breaking it down into smaller subtasks. These subtasks are, in turn, broken down further into more subtasks until, ultimately, the program is completely written. Top down design can be used in implementing programs in any language, including assembler. The essential idea of top down design is that you start writing a program by designing the MAINLINE program first. This program should consist almost exclusively of calls to subroutines. The next stage in the process is to design the subroutines which are called from the mainline program. Each of these subroutines should be like a mini–mainline program in itself consisting, perhaps, of several smaller subroutines. The word MODULE is generally used to refer to a subroutine and its subordinates which carry out one well–defined task for the main program.

MODULAR PROGRAMMING

The principle goal of MODULAR PROGRAMMING is to produce a program which is made up of well–defined MODULES which can be plugged into and pulled out of your program as needed without affecting other unrelated parts of the program. Each module should have only one entry point and only one exit point. With a modularly designed program, you can add modules or completely rewrite existing modules, without affecting the rest of the program as your needs change. Modular design is most important for business programs that are constantly mutating as the needs of the business change. As an individual user, however, you might use modular programming techniques to make your programs more compatible between machines by grouping machine dependent instructions into modules rather than having them sprinkled throughout the program. For instance, suppose you write a huge program on your C64 to compute your income tax and later decide that you want to implement the program on an IBM PC. If the program is well designed, you would probably only have to rewrite the INITIALIZATION module (which would open all files, set machine dependent parameters, etc.), the READ module (which reads from disk) and the WRITE module (which displays data to the screen or printer). Most other modules in the program could probably remain unchanged. You should note also that as tax laws change, you would only need to rewrite the modules affected by any given change from the current laws.

STRUCTURED PROGRAMMING

This bring us to the final point – structured programming. Everything I've mentioned so far has been merely conceptual. The ideas we've talked about have only to do with the DESIGN of a program. Structured programming is the IMPLEMENTATION of these ideas of top down design and modular programming but with some additional rules to make program logic easy to follow, thus making programs easier to debug and maintain. These additional rules include severely limiting the use of the GOTO statement, indenting program text so that the nesting of loops can be clearly seen and giving meaningful names to called program modules.

Clearly BASIC 2.0 is not well suited for the kind of programming described above. This is, of course, the reason that I'm presenting the program that accompanies this article. The new commands I am including here will allow you to write code which, for the most part, conforms to the rules of structured programming. The commands I am including are similar to those you will find in structured languages such as Pascal, COBOL, WATFIV and C. (They most closely resemble those found in Pascal and WATFIV). I chose these structures (WHILE/WEND, IF/THEN/ELSE, REPEAT/UNTIL, CALL/PROC, etc) because they so readily adapt themselves to the concepts of structured design. While I can't give you a course in how and when to use these structures, I will describe what each one does and leave you with examples of their usage. As you program using these commands, their usage will become natural to you and you will begin to appreciate their value.

If you already use a language such as Pascal on your 64 you will appreciate having these structures available for use in the interpretative environment of BASIC. The ease of debugging in the BASIC environment will cut your program development time down dramatically from what it was when you had to load an editor, compiler, linker and object module every time you needed to make a change in your program.

USING THE NEW COMMANDS

To use these new commands in your own programs, simply type in the BASIC Loader (LISTING 1) and activate the extended BASIC with SYS 49152. Here's a description of the commands available to you:

EDIT -- This command causes the C64 editor to view the SPACE as a valid character. This feature allows you to indent your program lines to clearly indicate nesting level (see the example program). This indentation is part of structured programming syntax. WARNING! When in EDIT mode you must eliminate spaces from any direct mode commands (except when in quotes) or a syntax error will result. Also, you should not type a space after the line number when entering program lines in EDIT mode. Edit mode is automatically turned off when you RUN a program.

KILL -- This command will manually turn off EDIT mode.

BASIC2 -- This command will disable the extended commands and return you to BASIC 2.0. Commands can be re-enabled with SYS 49152.

CALL/PROC -- The CALL statement works exactly like GOSUB except that you CALL the routine by name instead of using a line number. This allows you to assign meaningful names to program modules. The word PROC is used to mark the beginning of a subroutine (PROCedure) and must be the first statement on the line (No spaces may precede the word PROC - see error #6). A procedure is ended with the word RETURN.

```
10 FOR I=1 TO 10
20   CALL OUTPUT
30 NEXT I
40 END
50 :
60 PROC OUTPUT
70   PRINT "OUTPUT CALL #";I
80 RETURN
```

REPEAT/UNTIL -- This structure is quite similar to the FOR/NEXT loop except that instead of counting, like FOR/NEXT, the REPEAT command repeats a block of instructions until some condition becomes true.

```
10 REPEAT
15   X=X+1
20   PRINT X
30 UNTIL X>= 10
```

WHILE/WEND -- Clever use of the WHILE statement can eliminate a bunch of GOTOs from your code. It repeats a block of instructions WHILE some condition is true. The WHILE statement is similar to the REPEAT statement except the condition is checked BEFORE the loop is entered. REPEAT/UNTIL and FOR/NEXT loops, on the other hand, always execute the body of the loop at least once.

```
10 OPEN2,8,2,"TEST.FIL,S"
20 WHILE ST=0
30   GET#2,A$
40   PRINT A$;
50 WEND
```

The above code will read a sequential file and display it to the screen.

EXIT -- You probably know that it is illegal to exit a FOR/NEXT loop or a subroutine with a GOTO statement in BASIC 2.0. For the same reasons, it is illegal exit a REPEAT/UNTIL loop, a WHILE/WEND loop or a PROC with a GOTO statement. If a condition arises such that you must make an 'emergency' exit from one of the above structures, use the EXIT command. It works exactly like a GOTO except that it cleans up the stack before it GOes.

The EXIT command is intelligent. That is, you don't have to tell it what kind of loop or structure you are EXITing. It can figure out by itself whether it is EXITing a FOR/NEXT, REPEAT/UNTIL, GOSUB, PROC or WHILE/WEND structure. You can only EXIT one structure at a time. Thus, if you are nested three levels deep, you must only exit one level at a time (reason: For maximum flexibility the EXIT command only cleans one data set off the stack at a time).

```
10 X=1:INPUT "ENTER AN INTEGER";K
20 FOR I=K TO 1 STEP -1
30   X = X*K*RND(0)
40   IF ABS(X)>1E7 THEN EXIT 90
50   PRINT "X = ";X
60 NEXT
70 PRINT "FINAL RESULT:";X
80 END
90 PRINT "**** ERROR ****"
95 PRINT "NUMBER TOO LARGE"
99 PRINT "TRY AGAIN":GOTO10
```

IF/THEN/ELSE -- This simple one-line IF/THEN/ELSE allows you to choose between two actions based on a conditional statement.

```
IF ABS(X)<1 THEN CALL NEWTON:ELSE CALL GAUSS
```

That is read "If the absolute value of X is less than one, CALL NEWTON. Otherwise, CALL GAUSS."

ERROR MESSAGES

Since this extended BASIC is intended to be used in 'real-life' programming a full range of diagnostic error messages are included to aid in finding program bugs. Here is a list of the messages and the probable cause of each one.

#1 UNTIL WITHOUT REPEAT (Either you left out the REPEAT statement or you have improperly nested program structures. Could also be caused by improper use of the EXIT command.)

#2 WEND WITHOUT WHILE (see #1)

#3 WHILE WITHOUT WEND (Your program has less WEND statements than WHILE statements. See also #1.)

#4 MISSING LOGICAL EXPRESSION (The word WHILE or UNTIL was not followed by a logical expression.)

#5 NO STRUCTURE TO EXIT (Attempted to use the EXIT command outside of a structure. Could also indicate a problem in program flow. For example a GOTO whose target is inside of some structure.)

#6 PROCEDURE NOT FOUND (Either no such procedure exists or the word PROC was not the first word on the line. Make sure there is not an extra space at the beginning of the line before the word PROC)

#7 OUT OF MEMORY (This indicates either very deeply nested structures or problems in program flow. It usually occurs when structures are not properly exited.)

#8 PROC WITHOUT CALL (Means you left off the END statement before your procedure block. Also, could indicate problems in program flow.)

EXAMPLE PROGRAM

The example program at the end of this article doesn't do anything extremely novel and isn't the best example of structured programming but it DOES use all of the new commands included in this extended BASIC. When you run the program it will give you the option of reading a disk file or creating a disk file. If you choose to READ a file, you will simply be asked for a filename and the requested file will be displayed on your screen. If you choose to CREATE a file, you will be asked to type in some data after providing a filename. There are no restrictions on the type of data you can enter. Every key on the keyboard is considered valid data. When you have typed all you desire, simply hit the RETURN key TWICE to close the disk file.

TECHNICAL NOTES

Listing 2 contains the complete source code for this language extension. On my own system, I have implemented this program in tokenized form for maximum efficiency. The tokenizer and list routines were too long to include here so I rewrote the program in non-tokenized form. Whenever you see a machine instruction in the comment field in the source listing it means that in a tokenized version (single tokens) you would use that instruction instead of whatever instruction is shown on that line. If you are a TransBASIC user you should be able to easily convert these commands to TransBASIC modules (be sure to allow for the double token scheme of TB). I'm sure Nick Sullivan can help you if you have trouble with the conversion.

Because it involved some repetition of code, I neglected to implement a block IF/THEN/ELSE structure like

```
10 IF X>10 THEN
20    PRINT "X IS GREATER THAN 10"
30    PRINT "WHOOPIE!"
40 ELSE
50    PRINT "X LESS OR EQUAL 10"
60    PRINT "BIG DEAL"
70 ENDIF
```

If you would like to implement this structure yourself, here's an outline of what the assembly language routine corresponding to each keyword should do when executed.

IF -- Check condition. If true, business as usual. If false, search for ELSE and continue execution *after* the ELSE statement.

ELSE -- If an ELSE is *executed* it means you have gotten to the end of the IF block, so search for the ENDIF statement and continue execution after it. Note: The search routine will be exactly like the FNDWND routine in listing 3. This routine takes nesting into account, etc.

ENDIF -- ENDIF doesn't do anything other than mark the end of the IF statement. Just execute an RTS.

If anyone out there is really ambitious, a modified CALL/PROC routine which includes TRUE PARAMETER PASSING would be a nice addition to your extended BASIC. This is a project I've intended to take on for quite some time but if anyone would like to save me the trouble, I'd love to see it here in the Transactor.

Structured BASIC Extension: Loader

```
AG | 1000 rem loader for structured basic ext
JD | 1010 rem by frank digioia  6/2/86
BE | 1020 rem sys 49152 to activate
KH | 1030 :
AK | 1040 for adr=49152 to 50274:read ml
LO | 1050 cs=cs+ml:poke adr,ml:next
FC | 1060 if cs<>136922 thenprint " data error "
CK | 1070 :
HJ | 1080 data 169,  49, 141,   8,   3, 169, 192, 141
JA | 1090 data   9,   3, 169,  17, 160, 192,  76,  30
```

```
JB 1100 data 171,  62,  32,  83,  84,  82,  85,  67
PN 1110 data  84,  85,  82,  69,  68,  32,  67,  79
JP 1120 data  77,  77,  65,  78,  68,  83,  32,  69
NP 1130 data  78,  65,  66,  76,  69,  68,  46,  13
PG 1140 data   0,  32, 115,   0,  32,  67, 192,  76
DN 1150 data 174, 167,  76,  59, 169,  32,   5, 193
FL 1160 data  76, 180, 192, 201,  39, 240, 243, 201
CL 1170 data 139, 208,   3,  76,  44, 193, 201, 138
JC 1180 data 240, 235, 170,  48,  95, 160,  10, 140
DN 1190 data 186, 192, 201,  87, 208,   8, 160,   1
IF 1200 data 177, 122, 201, 128, 240,  52, 169,   0
HB 1210 data 141, 186, 192, 170, 168, 136, 200, 189
OA 1220 data 187, 192, 240,  64, 232, 209, 122, 208
HC 1230 data   2, 240, 243, 202, 189, 187, 192,  16
HM 1240 data   8,  41, 127, 209, 122, 240,  19, 208
CP 1250 data   8, 232, 189, 187, 192, 240,  37,  16
PC 1260 data 248, 232, 238, 186, 192, 160, 255,  76
AO 1270 data 110, 192, 152,  24, 101, 122, 133, 122
GE 1280 data 144,   2, 230, 123, 173, 186, 192,  10
MH 1290 data 170, 189, 235, 192,  72, 189, 234, 192
IE 1300 data  72,  76, 115,   0,  32, 121,   0,  76
DN 1310 data 237, 167,   0,  82,  69,  80,  69,  65
BK 1320 data 212,  85,  78,  84,  73, 204,  87,  72
CM 1330 data  73,  76, 197,  69,  88,  73, 212,  67
BO 1340 data  65,  76, 204,  80,  82,  79, 195,  69
PM 1350 data  76,  83, 197,  69,  68,  73, 212,  75
HB 1360 data  73,  76, 204,  66,  65,  83,  73,  67
GH 1370 data 178,   0, 129, 193, 155, 193, 217, 193
IO 1380 data 188, 194, 237, 194, 136, 195, 126, 193
HJ 1390 data 255, 192,   4, 193,   9, 193,  11, 194
GO 1400 data 169, 255, 133, 129,  96, 169,  32, 133
PP 1410 data 129,  96, 169, 228, 141,   8,   3, 169
FM 1420 data 167, 141,   9,   3, 169,  28, 160, 193
LE 1430 data  76,  30, 171,  96,  62,  32,  67,  77
IB 1440 data  68,  83,  32,  68,  73,  83,  65,  66
NK 1450 data  76,  69,  68,   0,  32, 115,   0,  32
HC 1460 data 158, 173,  32, 121,   0, 201, 137, 240
NO 1470 data   5, 169, 167,  32, 255, 174, 165,  97
GN 1480 data 208,   7,  32,  99, 193, 170, 208,  22
KF 1490 data  96,  32, 121,   0, 176,   3,  76, 160
DH 1500 data 168, 165, 122,  56, 233,   1, 133, 122
DN 1510 data 176,   2, 198, 123, 160,   0, 104, 104
AP 1520 data 108,   8,   3,  32,   6, 169,  72,  32
GC 1530 data 251, 168, 104, 240,  13, 162,   3,  32
NM 1540 data 115,   0, 221, 123, 193, 208, 236, 202
JB 1550 data  16, 245,  96,  69,  83,  76,  69,  76
PH 1560 data  59, 169, 169,   3,  32, 251, 163,  32
EI 1570 data 248, 168, 165, 123,  72, 165, 122,  72
BJ 1580 data 165,  58,  72, 165,  57,  72, 169, 231
NL 1590 data  72,  76, 174, 167,  32, 138, 163, 154
AC 1600 data 201, 231, 208,  40,  32, 121,   0, 240
NM 1610 data  44,  32, 158, 173, 186, 138,  24, 105
CH 1620 data   5, 170, 168, 165,  97, 208,  19, 162
LF 1630 data   1, 136, 185,   1,   1, 149, 122, 185
PF 1640 data 255,   0, 149,  57, 202,  16, 242,  76
HA 1650 data 174, 167, 154,  96, 169,   0,  44, 169
NH 1660 data   1,  44, 169,   2,  44, 169,   3,  76
FC 1670 data 189, 195,  32, 121,   0, 240, 246, 169
LM 1680 data   3,  32, 251, 163, 165, 122, 141, 185
PN 1690 data 194, 165, 123, 141, 186, 194,  32, 158
KC 1700 data 173, 165,  97, 208,   3,  76, 101, 194

BG 1710 data 173, 186, 194,  72, 173, 185, 194,  72
BC 1720 data 165,  58,  72, 165,  57,  72, 169, 235
JE 1730 data  72,  76, 174, 167,  32, 138, 163, 154
JE 1740 data 201, 235, 208, 187,  32, 121,   0, 208
PC 1750 data 185, 165, 123, 141, 186, 194, 165, 122
JF 1760 data 141, 185, 194, 165,  58, 141, 188, 194
NK 1770 data 165,  57, 141, 187, 194, 186, 138,  24
IO 1780 data 105,   5, 170, 142, 183, 194, 168, 162
LP 1790 data   1, 136, 185,   1,   1, 149, 122, 185
DO 1800 data 255,   0, 149,  57, 202,  16, 242,  32
HJ 1810 data 158, 173, 165,  97, 240,   3,  76, 174
PI 1820 data 167, 174, 183, 194, 154, 162,   1, 189
NK 1830 data 185, 194, 149, 122, 189, 187, 194, 149
FG 1840 data  57, 202,  16, 243,  96, 169,   0,  72
CI 1850 data  32, 248, 168,  32, 121,   0, 170, 240
HG 1860 data  18,  32, 115,   0, 170, 240,  12,  32
IH 1870 data 142, 195, 240,  34,  32, 145, 195, 240
MA 1880 data  35, 208, 229, 160,   2, 177, 122, 208
NJ 1890 data   3,  76, 210, 193, 200, 177, 122, 141
CN 1900 data 187, 194, 200, 177, 122, 141, 188, 194
OJ 1910 data  32, 251, 168,  76, 113, 194, 104, 240
NI 1920 data   9,  76, 104, 194, 169, 235,  72,  76
FN 1930 data 104, 194, 173, 187, 194, 133,  57, 173
MA 1940 data 188, 194, 133,  58,  76, 248, 168,   0
LF 1950 data   0,   0,   0,   0,   0, 104, 104, 104
EM 1960 data 201, 129, 240,  17, 201, 141, 240,  16
ED 1970 data 201, 231, 240,  12, 201, 235, 240,   8
PL 1980 data 169,   4,  76, 189, 195, 169,  19,  44
OE 1990 data 169,   6, 141, 184, 194, 186, 138,  24
LH 2000 data 109, 184, 194, 170, 154,  32, 121,   0
FD 2010 data  32, 160, 168,  76, 174, 167, 169,   3
FC 2020 data  32, 251, 163, 165, 123,  72, 165, 122
OD 2030 data  72, 165,  58,  72, 165,  57,  72, 169
IN 2040 data 141,  72,  32,  24, 195, 162,   1, 181
IH 2050 data 251, 149, 122, 181,  97, 149,  57, 202
CL 2060 data  16, 245,  32, 248, 168,  76, 174, 167
GG 2070 data 165,  43, 133, 253, 165,  44, 133, 254
EG 2080 data 165, 253, 133, 251, 165, 254, 133, 252
LE 2090 data 160,   1, 177, 251, 208,   5, 169,   5
BN 2100 data  76, 189, 195, 133, 254, 136, 177, 251
LL 2110 data 133, 253, 160,   4, 177, 251,  32, 159
OF 2120 data 195, 208, 221, 160,   3, 177, 251, 133
ON 2130 data  98, 136, 177, 251, 133,  97, 160,   7
CG 2140 data 200, 177, 251, 201,  32, 240, 249, 152
GP 2150 data  24, 101, 251, 133, 251, 144,   2, 230
JG 2160 data 252, 160, 255, 200, 177, 122, 240,  24
KF 2170 data 201,  58, 240,  20, 201,  32, 208,   9
GL 2180 data 230, 122, 208,   2, 230, 123,  76, 100
BO 2190 data 195, 209, 251, 240, 230,  76,  32, 195
AB 2200 data 177, 251, 240,   4, 201,  58, 208, 152
OF 2210 data  96, 169,   6,  76, 189, 195, 162,   4
MB 2220 data  44, 162,   7, 164, 122, 132, 251, 164
OM 2230 data 123, 132, 252, 160, 255, 208,   4, 162
FN 2240 data 255, 160,   3, 200, 232, 189, 175, 195
CN 2250 data 240,   4, 209, 251, 240, 245,  96,  80
BB 2260 data  82,  79,  67,   0,  87, 128,   0,  87
GG 2270 data  72,  73,  76,  69,   0,  10, 170, 189
EH 2280 data 202, 195, 133,  34, 189, 203, 195,  76
AK 2290 data  69, 164, 216, 195, 236, 195, 254, 195
OG 2300 data  16, 196,  42, 196,  62, 196,  81, 196
DJ 2310 data  85,  78,  84,  73,  76,  32,  87,  73
```

```
JI 2320 data  84,  72,  79,  85,  84,  32,  82,  69
NA 2330 data  80,  69,  65, 212,  87,  69,  78,  68
FK 2340 data  32,  87,  73,  84,  72,  79,  85,  84
KC 2350 data  32,  87,  72,  73,  76, 197,  87,  72
GL 2360 data  73,  76,  69,  32,  87,  73,  84,  72
BE 2370 data  79,  85,  84,  32,  87,  69,  78, 196
BM 2380 data  77,  73,  83,  83,  73,  78,  71,  32
MN 2390 data  76,  79,  71,  73,  67,  65,  76,  32
CA 2400 data  69,  88,  80,  82,  69,  83,  83,  73
OF 2410 data  79, 206,  78,  79,  32,  83,  84,  82
GA 2420 data  85,  67,  84,  85,  82,  69,  32,  84
FF 2430 data  79,  32,  69,  88,  73, 212,  80,  82
AC 2440 data  79,  67,  69,  68,  85,  82,  69,  32
AC 2450 data  78,  79,  84,  32,  70,  79,  85,  78
EH 2460 data 196,  80,  82,  79,  67,  32,  87,  73
DC 2470 data  84,  72,  79,  85,  84,  32,  67,  65
PK 2480 data  76, 204,   0
```

Structured BASIC Extension: Example Program

```
CL 100 '
FI 110 ' structured demo -- frank digioia
GM 120 '
CA 130 call prompt
MF 140 while a$<>"x"
MC 150    if a$="c" then call create:else call read
AC 160    call prompt
IN 170 wend
EL 180 end
CD 190 :
FC 200 proc create
ML 210 call get-name
KI 220 open2,8,2,f$+",w"
BA 230 print"enter data. . ."
LB 240 call get-key
AH 250 repeat
EM 260    print a$;
OL 270    print#2,a$;
OI 280    repeat
NE 290       call get-key
MO 300       print a$;
GO 310       print#2,a$;
JF 320    until a$ = chr$(13)
FH 330    call get-key
NG 340 until a$=chr$(13)
AB 350 print#2:close2
EI 360 return
GO 370 :
PJ 380 proc read
AH 390 call get-name
IH 400 open2,8,2,f$
FH 410 while st=0
BD 420    get#2,a$
OG 430    print a$;
GO 440 wend
IJ 450 close2
IO 460 return
KE 470 :
BE 480 proc get-valid-key
AG 490 repeat
PB 500    call get-key
GF 510 until a$="c" or a$="d" or a$="x"
EC 520 return
GI 530 :
HJ 540 proc get-key
CE 550 a$=""
PF 560 while a$=""
PM 570    get a$
CH 580 wend
KG 590 return
MM 600 :
MJ 610 proc get-name
CJ 620 f$=""
EL 630 while f$=""
FF 640    input"filename";f$
IL 650 wend
AL 660 return
CB 670 :
IH 680 proc prompt
MP 690 print"[rvs] type c to create a data file"
MO 700 print"type d to display a data file"
II 710 print"type x to exit program"
HP 720 print"[rvs] note: when creating a data file"
HF 730 print"hit <return> twice to end input"
FP 740 call get-valid-key
KA 750 return
```

Structured BASIC Extension: PAL Source

```
OF 1000 ;
MF 1010 ;structured programming (parser)
JK 1020 ;by frank e. digioia
OM 1030 ;11/12/85
GI 1040 ;
HM 1050 *       =   $c000     ;convenient start
KJ 1060 ;
OJ 1070 chrget  =   $0073     ;get byte of text
JG 1080 chrgot  =   $0079     ;get same byte
LC 1090 igone   =   $0308     ;evaluation vector
CM 1100 ;
MO 1110 init    =   *         ;initialize routine
PK 1120         lda #<struct
NO 1130         sta igone
PL 1140         lda #>struct
NA 1150         sta igone+1
FI 1160         lda #<note
LO 1170         ldy #>note
MM 1180         jmp $ab1e
MB 1190 ;
NO 1200 note    .asc ">structured commands"
KN 1210         .asc " enabled." : .byte $0d,$00
KD 1220 ;
DA 1230 struct  =   *
PL 1240         jsr chrget    ;get a byte of text
KM 1250         jsr chkout    ;" structured command?
PO 1260         jmp $a7ae     ;intepreter loop
MG 1270 ;
JC 1280 rem     jmp $a93b     ;rem command
AI 1290 ;
AG 1300 newrun  jsr kill      ;kill edit mode
OA 1310         jmp basic     ;give to basic
OJ 1320 ;
GE 1330 chkout  cmp #$27      ;" single quote?
IN 1340         beq rem       ;classy rem
CA 1350         cmp #$8b      ;can't have new cmds
DB 1360         bne *+5       ;without a new if
AK 1370         jmp if
MC 1380         cmp #$8a      ;'run' token
HC 1390         beq newrun    ;end edit and run
DF 1400         tax           ;set flags
PD 1410         bmi basic     ;token/give to basic
CA 1420 ;
OP 1430         ldy #$0a      ;check on 'wend'
AF 1440         sty count     ;point to 'wend'
MN 1450         cmp #"w"      ;" current char = 'w'?
BF 1460         bne setup     ;no/not wend
HJ 1470         ldy #$01      ;yes/check next char
IB 1480         lda ($7a),y   ;next byte of text
BK 1490         cmp #$80      ;" 'end'?
CP 1500         beq exec      ;yes/execute wend
MF 1510 ;
MI 1520 setup   lda #$00      ;clear all regs
PI 1530         sta count     ;and keyword counter
PM 1540         tax
NN 1550         tay
KC 1560         dey           ;pre-loop decrement
IJ 1570 ;
ON 1580 loop    iny           ;incr text index
NO 1590         lda table,x   ;get table byte
ML 1600         beq basic     ;end of table
FO 1610         inx           ;incr table pointer
MF 1620         cmp ($7a),y   ;cmpare with text
OL 1630         bne next      ;find next word
BM 1640         beq loop      ;match/keep looking
IO 1650 ;
EC 1660 next    dex           ;bump .x down once
MI 1670         lda table,x   ;" end of table word?
EN 1680         bpl find      ;no/find end of word
CH 1690         and #$7f      ;yes/mask flag
CC 1700         cmp ($7a),y   ;" is it a match?
II 1710         beq exec      ;hooray!!!
JM 1720         bne x1        ;go back for more
ID 1730 ;
BD 1740 find    inx           ;find end of word
HA 1750         lda table,x   ;look for negative
MF 1760         beq basic     ;end of table
IA 1770         bpl find      ;keep looking
KG 1780 ;
EI 1790 x1      inx           ;point to next word
FO 1800         inc count     ;word # in table
PA 1810         ldy #$ff      ;reset text index
EM 1820         jmp loop      ;search some more
MJ 1830 ;
ME 1840 exec    =   *         ;execution routine
PN 1850         tya           ;update text pointer
KL 1860         clc
```

```
NP 1870          adc  $7a
DF 1880          sta  $7a
HF 1890          bcc  *+4
ME 1900          inc  $7b
MO 1910 ;
FK 1920          lda  count     ;get offset in table
NE 1930          asl  a         ;multiply by two
BC 1940          tax            ;use as index
ME 1950          lda  adrtab+1  ;hi byte adr
KD 1960          pha            ;as return adr hi
GF 1970          lda  adrtab,x  ;lo byte adr
EG 1980          pha            ;as return adr lo
PJ 1990          jmp  chrget    ;execute routine
GE 2000 ;
EO 2010 basic    jsr  chrgot    ;reset flags
CM 2020          jmp  $a7ed     ;give it to basic
EG 2030 ;
HD 2040 count    .byte $00
IH 2050 ;
HB 2060 table    .asc "repea" : .byte $d4
OK 2070          .asc "unti" : .byte $cc
NH 2080          .asc "whil" : .byte $c5
GA 2090          .asc "exi" : .byte $d4
EP 2100          .asc "cal" : .byte $cc
GC 2110          .asc "pro" : .byte $c3
HC 2120          .asc "els" : .byte $c5
GA 2130          .asc "edi" : .byte $d4
ED 2140          .asc "kil" : .byte $cc
AH 2150          .asc "basic" : .byte $b2,$00
GO 2160 ;
KB 2170 adrtab   .word repeat-1,until-1
DO 2180          .word while-1,exit-1,call-1
AJ 2190          .word xproc-1,else-1,edit-1,kill-1
MD 2200          .word basic2-1,wend-1
IB 2210 ;
IC 2220 ;edit mode commands
MC 2230 ;
OE 2240 edit     lda  #$ff      ;ignore pi symbol
FA 2250          sta  $81       ;alter chrget
EN 2260          rts            ;that's it!
EF 2270 ;
NC 2280 kill     lda  #$20      ;ignore spaces
GG 2290          sta  $81       ;fix chrget
IO 2300          rts
MH 2310 ;
MN 2320 basic2   lda  #$e4      ;fix igone vector
NJ 2330          sta  igone
GJ 2340          lda  #$a7
NL 2350          sta  igone+1
DG 2360          lda  #<note2   ;notify user
NM 2370          ldy  #>note2
MH 2380          jmp  $ab1e
CE 2390          rts
IP 2400 note2    .asc "> cmds disabled" : .byte $00
AO 2410 ;
FE 2420 ;structured programming module
LC 2430 ;by frank e. digioia
FF 2440 ;11/23/85
IA 2450 ;
JD 2460 ;tokens for lookups & cmp's
MB 2470 ;
PA 2480 whltok   =    $eb
NA 2490 wndtok   =    $ec
KA 2500 reptok   =    $e7
JA 2510 gosubs   =    $8d
KB 2520 for      =    $81
AB 2530 proc     =    $e5
CG 2540 ;
II 2550 stack    =    $0100    ;6510 stack area
JL 2560 frmevl   =    $ad9e    ;evaluate formula
CE 2570 getptr   =    $a38a    ;pntr to stack id
EA 2580 chkstk   =    $a3fb    ;check stack space
EJ 2590 ;
EA 2600 if       =    *
PC 2610          jsr  chrget    ;get next byte
AA 2620          jsr  $ad9e     ;evaluate expression
MH 2630          jsr  $0079     ;get last char
HD 2640          cmp  #$89      ;" 'goto' token?
GE 2650          beq  chkexp    ;yeah/check result
BD 2660          lda  #$a7      ;'then' token
PE 2670          jsr  $aeff     ;check on 'then'
DD 2680 chkexp   lda  $61       ;" expression true?
GA 2690          bne  doit      ;yes/execute cmd
NB 2700          jsr  fndels    ;no/look for 'else'
AJ 2710          tax            ;" eoln?
NB 2720          bne  cmmd      ;no/do else clause
AA 2730          rts            ;yes/return to interp
KC 2740 ;
BA 2750 doit     jsr  chrgot    ;get last char
KB 2760          bcs  decptr    ;not digit/execute it
OL 2770          jmp  $a8a0     ;digit/execute goto
CF 2780 ;
GL 2790 decptr   lda  $7a       ;decrement txtptr
BH 2800          sec
BD 2810          sbc  #$01
PP 2820          sta  $7a
DE 2830          bcs  *+4
DN 2840          dec  $7b
EB 2850          ldy  #$00      ;clear .y for update
CK 2860 ;
OL 2870 cmmd     pla            ;clear return address
IM 2880          pla
OC 2890          jmp  ($0308)   ;execute via vector
KM 2900 ;
FD 2910 fndels   jsr  $a906     ;find next stmt
PE 2920          pha            ;save byte
CL 2930          jsr  $a8fb     ;update txtptr
MC 2940          pla            ;get byte back
LL 2950          beq  noelse    ;" end of line?
BD 2960          ldx  #$03      ;compare 4 byte
AD 2970 chkels   jsr  chrget    ;get a byte
BK 2980          cmp  esle,x    ;comare bkwrd
KD 2990          bne  fndels    ;no/next stmt
CD 3000          dex            ;bump index
JE 3010          bpl  chkels    ;keep checking
MG 3020 noelse   rts
ME 3030 ;
DB 3040 esle     .asc "esle"
AG 3050 ;
KO 3060 else     jmp  $a93b     ;do a rem
EH 3070 ;
CP 3080 repeat   =    *
MK 3090          lda  #$03      ;need 6 bytes
LN 3100          jsr  chkstk    ;check stack space
GA 3110          jsr  $a8f8     ;point next statement
FO 3120          lda  $7b       ;save text pointer
GL 3130          pha
BA 3140          lda  $7a
KM 3150          pha
OA 3160          lda  $3a       ;save line number
ON 3170          pha
JA 3180          lda  $39
CP 3190          pha
IP 3200          lda  #reptok
GA 3210          pha
JG 3220          jmp  $a7ae     ;interpreter loop
EB 3230 ;
DD 3240 until    =    *
BO 3250          jsr  getptr    ;find id on stack
EM 3260          txs            ;replace pointer
PK 3270          cmp  #reptok   ;" repeat id?
FE 3280          bne  uerr1     ;'missing repeat'
OK 3290          jsr  chrgot    ;" condition present?
PC 3300          beq  nocond    ;'missing cond.'
OL 3310          jsr  frmevl    ;evaluate expression
HH 3320          tsx            ;get stack pointer
JA 3330          txa            ;place in .a
CI 3340          clc
GJ 3350          adc  #$05      ;backup 5 on stack
LO 3360          tax
JP 3370          tay
KK 3380 ;
KB 3390          lda  $61       ;check result (t/f)
IP 3400          bne  utrue     ;true/fix stack
IM 3410 ;
ID 3420          ldx  #01       ;false/copy data from
EF 3430 getdat   dey            ;stack into program
EE 3440          lda  stack+1,y ;pointer & curlin
LC 3450          sta  $7a,x     ;to continue execution
KA 3460          lda  stack-1,y ;at top of loop.
BH 3470          sta  $39,x
PE 3480          dex
CI 3490          bpl  getdat
BI 3500          jmp  $a7ae     ;interpreter loop
MC 3510 ;
GC 3520 utrue    txs            ;update stack pointer
GL 3530          rts
KE 3540 ;
KC 3550 uerr1    lda  #$00
DM 3560          .byte $2c
DE 3570 werr1    lda  #$01
HN 3580          .byte $2c
KF 3590 werr2    lda  #$02
LO 3600          .byte $2c
CK 3610 nocond   lda  #$03
OC 3620          jmp  error     ;print error msg
EK 3630 ;
OH 3640 while    =    *
GB 3650          jsr  chrgot    ;" condition present?
NN 3660          beq  nocond    ;no/error mesg
AP 3670          lda  #$03      ;need 6 bytes
PB 3680          jsr  chkstk    ;check stack space
NN 3690          lda  $7a       ;save pointer to
NJ 3700          sta  t1        ;the conditional
CE 3710          lda  $7b       ;expression for
JE 3720          sta  t2        ;later use.
CG 3730          jsr  frmevl    ;evaluate expression
ND 3740          lda  $61       ;" true or false?
II 3750          bne  wtrue     ;true/load up stack
DA 3760          jmp  fndwnd    ;false/find wend
AD 3770 ;
OL 3780 wtrue    lda  t2        ;save pointer to
AI 3790          pha            ;the logical
KN 3800          lda  t1        ;expression on
IB 3810          pha            ;stack
CK 3820          lda  $3a       ;save line number
AM 3830          pha            ;on stack
NJ 3840          lda  $39
GI 3850          pha
CB 3860          lda  #whltok   ;save id for while
IO 3870          pha            ;on stack
HH 3880          jmp  $a7ae
IK 3890 ;
JJ 3900 wend     jsr  getptr    ;find id on stack
PO 3910          txs            ;update pointer
HL 3920          cmp  #whltok   ;" id for while?
IB 3930          bne  werr1     ;'missing while'
DB 3940          jsr  chrgot    ;" end of statement?
CI 3950          bne  werr2     ;no/something wrong
OO 3960 ;
HD 3970          lda  $7b       ;save text pointer
KO 3980          sta  t2
DF 3990          lda  $7a
MP 4000          sta  t1
PF 4010          lda  $3a
EN 4020          sta  ll2
LF 4030          lda  $39
FO 4040          sta  ll1
IE 4050 ;
LF 4060          tsx            ;get stack pointer
NO 4070          txa            ;place in .a
GG 4080          clc
KH 4090          adc  #$05      ;back up 5 on stack
PM 4100          tax
CM 4110          stx  stkptr    ;store stack pointer
HO 4120          tay
IJ 4130 ;
PO 4140          ldx  #$01      ;get adr of while
MH 4150 whldat   dey            ;condition into
OM 4160          lda  stack+1,y ;$7a/$7b and line
AI 4170          sta  $7a,x     ;number into $39/$3a
LA 4180          lda  stack-1,y ;for frmevl to use
BE 4190          sta  $39,x
PB 4200          dex
KN 4210          bpl  whldat
CP 4220 ;
GF 4230          jsr  frmevl    ;evaluate expression
BD 4240          lda  $61       ;" true or false?
EO 4250          beq  wfalse
HO 4260          jmp  $a7ae     ;true/cont execution
EC 4270 ;
JJ 4280 wfalse   ldx  stkptr
FO 4290          txs            ;update stack pointer
PE 4300          ldx  #$01
JK 4310 wfill    lda  t1,x      ;replace text pntr
DO 4320          sta  $7a,x
OM 4330          lda  ll1,x     ;replace line number
HN 4340          sta  $39,x
FL 4350          dex
GO 4360          bpl  wfill
DB 4370          rts            ;continue execution
CJ 4380 ;
DM 4390 fndwnd   =    *         ;find wend statement
DF 4400          lda  #$00
DI 4410          pha            ;set flag on stack
BJ 4420 wsrch    jsr  $a8f8     ;find next stment
CF 4430          jsr  chrgot    ;" end of line?
DC 4440          tax
```

```
MF 4450          beq  eoln1      ;yes/deal with it
LJ 4460 xx       jsr  chrget     ;get next byte
LF 4470          tax             ;" end of line?
KH 4480          beq  eoln1      ;yes/deal with it
CK 4490          jsr  chkwnd     ;cmp #wndtok
BL 4500          beq  xwend
KM 4510          jsr  chkwhl     ;cmp #whltok
HB 4520          beq  xwhile
LH 4530          bne  wsrch
CD 4540 ;
IM 4550 eoln1    ldy  #$02       ;check for end text
OO 4560          lda  ($7a),y    ;" link hi = 0?
FP 4570          bne  *+5        ;no/continue search
PL 4580          jmp  werr2      ;yes/missing wend
NB 4590          iny             ;no/get line#
LL 4600          lda  ($7a),y    ;save line #
PB 4610          sta  ll1
MO 4620          iny
DC 4630          lda  ($7a),y
AE 4640          sta  ll2
LN 4650          jsr  $a8fb      ;update text pointer
AD 4660          jmp  xx         ;do search
EL 4670 ;
IE 4680 xwend    pla             ;check flag
CG 4690          beq  wndfnd     ;found it!!!
OF 4700          jmp  wsrch
MN 4710 ;
IK 4720 xwhile   lda  #whltok
GP 4730          pha
GI 4740          jmp  wsrch
EA 4750 ;
MI 4760 wndfnd   lda  ll1        ;load line #
NH 4770          sta  $39
OI 4780          lda  ll2
JK 4790          sta  $3a
EC 4800          jmp  $a8f8      ;find next statement
AE 4810 ;
AG 4820 stkptr   .byte $00
KC 4830 incrst   .byte $00
DP 4840 t1       .byte $00
AA 4850 t2       .byte $00
GC 4860 ll1      .byte $00
ED 4870 ll2      .byte $00
GI 4880 ;
MI 4890 exit     =    *
BI 4900          pla             ;find id on stack
GL 4910          pla
AM 4920          pla
KJ 4930          cmp  #for       ;" for command?
OD 4940          beq  getinc     ;get # of bytes
AJ 4950          cmp  #gosubs    ;" gosub command?
AD 4960          beq  getinc+3
HC 4970          cmp  #reptok
EE 4980          beq  getinc+3
OD 4990          cmp  #whltok
IF 5000          beq  getinc+3
JN 5010          lda  #$04       ;error number 4
IC 5020          jmp  error      ;'nothing to exit'
MB 5030 ;
GP 5040 getinc   lda  #$13       ;19 bytes on stack
JC 5050          .byte $2c       ;skip next instr.
AA 5060          lda  #$06       ;6 bytes on stack
MB 5070          sta  incrst     ;incr for stkptr
HF 5080          tsx             ;get stack pointer
CB 5090          txa             ;put in .a for add
CG 5100          clc
CH 5110          adc  incrst     ;increase stkptr
FF 5120          tax             ;replace it
OO 5130          txs             ;stack clean!
CC 5140          jsr  chrgot     ;get last char.
OB 5150          jsr  $a8a0      ;goto command
NP 5160          jmp  $a7ae      ;interpreter loop
IK 5170 ;
JG 5180 call     =    *
AO 5190          lda  #$03       ;need 6 bytes
PA 5200          jsr  chkstk     ;check stack space
PA 5210          lda  $7b        ;save text pointer
AO 5220          pha
LC 5230          lda  $7a
EP 5240          pha
ID 5250          lda  $3a        ;save line number
IA 5260          pha
DD 5270          lda  $39
MB 5280          pha
PF 5290          lda  #$8d       ;id for gosub
AD 5300          pha
```

```
ED 5310 ;
HC 5320          jsr  fndprc     ;find procedure adr
BF 5330          ldx  #$01       ;use .x as index
DM 5340 z        lda  $fb,x
NB 5350          sta  $7a,x      ;update text pointer
DI 5360          lda  $61,x
EB 5370          sta  $39,x      ;update line number
LL 5380          dex
MA 5390          bpl  z
OI 5400 ;
FP 5410          jsr  $a8f8      ;find next command
NO 5420          jmp  $a7ae      ;to interpreter loop
MK 5430 ;
NO 5440 fndprc   =    *          ;find procedure
MI 5450          lda  $2b        ;start of basic
KK 5460          sta  $fd        ;as pointer
HB 5470          lda  $2c
NI 5480          sta  $fe
IO 5490 ;
BG 5500 srchlp   lda  $fd        ;update link pntr
CK 5510          sta  $fb
HH 5520          lda  $fe
JL 5530          sta  $fc
KB 5540 ;
DD 5550          ldy  #$01       ;use .y as index
OC 5560          lda  ($fb),y    ;hi byte next line
ID 5570 ;
NC 5580          bne  *+7        ;" end of text?
PJ 5590          lda  #$05       ;yes/error number 5
KP 5600          jmp  error      ;'proc not found'
AG 5610 ;
NC 5620          sta  $fe        ;save next adr hi
FE 5630          dey             ;bump pointer
BI 5640          lda  ($fb),y    ;get next adr lo
CE 5650          sta  $fd        ;save it
CJ 5660 ;
IK 5670          ldy  #$04       ;point to 1st byte
DJ 5680          lda  ($fb),y    ;get the byte
HO 5690          jsr  chkprc     ;cmp #proc
FK 5700          bne  srchlp     ;no/try next line
EM 5710 ;
HC 5720          ldy  #$03       ;yes/get line #
AH 5730          lda  ($fb),y    ;get hi byte
GE 5740          sta  $62        ;save it
BD 5750          dey
IK 5760          lda  ($fb),y    ;get lo byte
BG 5770          sta  $61        ;save it
KA 5780 ;
LA 5790          ldy  #$07       ;ldy #$04
ON 5800 xspc     iny             ;skip leading spaces
DP 5810          lda  ($fb),y    ;get byte of name
IG 5820          cmp  #" "       ;" space?
HF 5830          beq  xspc
GE 5840 ;
BJ 5850          tya             ;get offset in .a
KF 5860          clc
OM 5870          adc  $fb        ;update our txtptr
LF 5880          sta  $fb        ;to first byte of
BA 5890          bcc  *+4        ;procedure name
NA 5900          inc  $fc
MI 5910 ;
AB 5920          ldy  #$ff       ;set .y = -1
DH 5930 compar   iny             ;update index
HE 5940 chktxt   lda  ($7a),y    ;byte of name
PP 5950          beq  chklst     ;end of exec name
JG 5960          cmp  #":"       ;" end of exec name?
JL 5970          beq  chklst     ;check end procname
IA 5980          cmp  #" "       ;" space?
BL 5990          bne  chknam     ;no/check proc name
HB 6000          inc  $7a        ;forget spaces
IJ 6010          bne  *+4
EG 6020          inc  $7b
NE 6030          jmp  chktxt
OA 6040 ;
CO 6050 chknam   cmp  ($fb),y    ;cmp proc name
CA 6060          beq  compar     ;match/keep checking
KB 6070          jmp  srchlp     ;no/find next proc
GD 6080 ;
DO 6090 chklst   lda  ($fb),y    ;" end procname?
NA 6100          beq  *+6
IC 6110          cmp  #":"
GD 6120          bne  srchlp
ON 6130          rts
CH 6140 ;
LD 6150 xproc    lda  #$06       ;error number 6
KG 6160          jmp  error
```

```
AJ 6170 ;
FD 6180 ;this routine may be omitted if
MC 6190 ;tokens are used (see article).
OK 6200 ;
HG 6210 chkwnd   ldx  #$04       ;offset for wend
NI 6220          byte $2c        ;skip next instr
DK 6230 chkwhl   ldx  #$07       ;offset to while
HE 6240          ldy  $7a        ;copy text pointer
LG 6250          sty  $fb        ;to $fb/$fc
EJ 6260          ldy  $7b
NP 6270          sty  $fc
HM 6280          ldy  #$ff       ;pre-loop index
PI 6290          bne  chkx       ;do the check
CB 6300 ;
LA 6310 chkprc   ldx  #$ff       ;offset for proc
AO 6320          ldy  #$03       ;pre-loop
AD 6330 ;
LF 6340 chkx     iny             ;compare loop
GD 6350          inx             ;bump pointer
AM 6360          lda  name,x     ;get byte of name
OE 6370          beq  xit        ;" end of name?
NN 6380          cmp  ($fb),y    ;compare to text
PA 6390          beq  chkx       ;match, keep on
OM 6400 xit      rts
AI 6410 ;
HC 6420 name     .asc "proc" : .byte $00
BE 6430          .asc "w" : .byte $80,$00
HF 6440          .asc "while" : .byte $00
IK 6450 ;
LA 6460 ;error processor -- prints error
IA 6470 ;messages and passes control to
JN 6480 ;rom error routines
AN 6490 ;
JD 6500 ;frank e. digioia
JE 6510 ;12/17/85
OO 6520 ;
GJ 6530 error    asl  a          ;mult err# by 2
JB 6540          tax             ;use as index
IF 6550          lda  errmsg,x   ;get mesg address
EG 6560          sta  $22
EK 6570          lda  errmsg+1,x
PB 6580          jmp  $a445      ;process error
ED 6590 ;
KA 6600 errmsg   .word u1msg,w1msg,w2msg
OP 6610          .word ncmsg,nemsg,npmsg,nocall
CF 6620 ;
BG 6630 u1msg    asc "until without repea" : .byte $d4
ID 6640 w1msg    asc "wend without whil" : .byte $c5
BC 6650 w2msg    asc "while without wen" : .byte $c4
FP 6660 ncmsg    asc "missing logical expressio"
         : .byte $ce
CJ 6670 nemsg    asc "no structure to exi" : .byte $d4
BK 6680 npmsg    asc "procedure not foun" : .byte $c4
PK 6690 nocall   asc "proc without cal" : .byte $cc
CK 6700 ;
CB 6710 .end
```

Blazin' Forth

Scott Ballantyne
New York, New York

Copyright (c) 1986, by Scott Ballantyne.

Everything you wanted to know about Forth (but were afraid to ask).

This article contains a description of how a threaded–code Forth compiler works, with specific reference to Blazin' Forth. You'll find this information useful if you are interested in the particulars of how Forth compilers work, or are interested in improving or changing one. Specifically, I wrote the following as an aid to people who might be trying to understand the source to Blazin' Forth, and it should be considered part of the documentation for the source files to the system.

To understand this document, you will need a decent knowledge of Forth. An understanding of pointers won't hurt any either.

You don't need to know machine language to understand this article, but you will, of course, need it to understand the actual source.

I have attempted to provide sample code in hi–level forth that illustrates the routines involved. This code is similar, but not exactly the same, as the actual machine level routines in the actual compiler. In particular, you should not expect these routines to actually work if you type them into a forth system in an attempt to build a "Forth in Forth". They are provided to add clarity, and that is their only function.

What is a Virtual Machine?

A Virtual Machine is a creation, in software, of a piece of hardware. Note that this hardware does not actually have to exist – it is only within the last year that an actual hardware Forth computer has been built. Forth has been around a lot longer than that.

All high level languages are essentially virtual machines, since they implement instructions which are not part of the actual hardware CPU. As an example, Forth uses two stacks, a parameter stack, and a return stack. On an actual hardware Forth computer, the built–in machine language of the computer would contain instructions for manipulating each stack, and the pointers to the bottom of each stack. On the 6502, there is actually only one stack – one or the other of the Forth stacks must be emulated using software routines. So you could say that one stack is a hardware stack, and the other stack is a Virtual stack.

As another example, the function call mechanism (how the actual functions, procedures, or words are eventually caused to execute) of a high level language is rarely directly supported by hardware instructions. On the majority of CPU's in use on personal computers today, the only real function call mechanism implemented in the hardware is the subroutine call (usually referred to as a JSR or CALL instruction). This instruction usually only saves the return address automatically. Any other information or any other method of invoking a subroutine must be done by software that, essentially, is a software (or virtual) function call, as opposed to the hardware function call and return. This is particularly true of threaded code Forth, and the routine which implements this function call mechanism is called NEXT. Understanding NEXT is the key to understanding the functioning of the Forth compiler at its lowest level.

Introducing NEXT.

To understand how NEXT (and the various Forth machine registers that NEXT uses to do its thing) works, let's first take a quick look at the structure of a compiled higher level Forth word. It is essentially a list of addresses:

```
: EXAMPLE  W1 W2 W3 ;

        ( Standard Forth Header goes here)
            Address of W1
            Address of W2
            Address of W3
            Address of EXIT ( ; )
```

NEXT uses several auxiliary registers to keep track of where the user program is. On a CPU with many registers, these would be kept in a selected CPU register. On the 6502, which has only 4 user–accessible registers, these are maintained as virtual registers (page zero locations are used for greater speed). One of these virtual registers is called the Interpreter Pointer, or IP for short, and it is responsible for keeping track of the progress of the current program. When NEXT is entered, in the course of running a program, the IP will be pointed at the word we want to execute. NEXT does some stuff (to be described in a moment) to cause this word to begin to execute, but before transferring control to this word, it moves the IP ahead to the next word's address, so it will know what word to run the next time it is called.

Here is a sample execution of EXAMPLE, given above:

```
IP → Address of W1 ( NEXT executes W1, first moving IP
     ahead to W2)
IP → Address of W2 ( NEXT executes W2, first moving IP
     ahead to W3)
IP → Address of W3 ( NEXT executes W3, first moving IP
     ahead to EXIT)
IP → Address of EXIT
```

I like to think of the IP as a kind of Address Slider, that can be moved ahead or behind to direct the flow of execution of the current program.

Ultimately, of course, NEXT must cause machine language instructions to be executed, which essentially means changing the hardware program counter of the CPU to point to the appropriate batch of instructions to be executed. It does this using another virtual register called the Current Word Pointer, or W for short. To understand this portion of NEXT, we have to clear up exactly what we mean by "Address of Word" in the above discussion.

The individual members of the list that makes up a Forth definition's executable body are the addresses of the code field in the header of the compiled word. (These "addresses of code fields" will be referred to as "the execution address" of a word in the rest of this document. When the term "code field" is used, the reference will be to the actual code field portion of a dictionary header. Or, at least, that is how I am going to try to use these terms.) This execution address (as you may recall) itself stores an address which points to machine level (assembly language) instructions. It is these instructions that NEXT causes the CPU to execute, by forcing the CPU's program counter to the address stored at the execution address of that word. So we have a couple of levels of indirection here:

> The IP points to a location which holds the execution address of a word. The execution address pointed to by the location pointed to by the IP points to executable machine language instructions.

So the full story of NEXT is as follows:

1) NEXT retrieves the value stored at the address in the IP.
2) It saves this value (the execution address of a word) in W (the current word pointer).
3) It then moves the IP ahead to point at the next word to be executed.
4) Finally, it forces the hardware program counter to the value stored at the address in W, which causes machine level instructions to execute.

Here is an example of the full execution of a forth word. Let's make up some example addresses for our example execution:

Address	Contents	Description
$A000	($0600)	W1's execution address is $A000, and contains $0600.
$B000	($0700)	W2's execution address is $B000, and contains $0700.
$C000	($0800)	W3's execution address is $C000, and contains $0800.
$0900	($0880)	EXIT's execution address is $0900, and contains $0880.

And here is how the compiled EXAMPLE word from earlier looks – let's say that the body of example starts at $E000:

Address	Contents	Description
$E000	($A000)	Compiled W1
$E002	($B000)	Compiled W2
$E004	($C000)	Compiled W3
$E006	($0900)	Compiled EXIT.

So, at the entry to NEXT, the IP will contain $E000.

NEXT fetches the address stored here, and stuffs it into W, so W will now contain $A000, which is the execution address of W1.

NEXT now increments the IP to point at the next word, so the IP will contain $E002.

Finally, NEXT forces the program counter of the CPU to the address stored in the address stored in W. So here's a quickie quiz – what will be the address in the hardware program counter?

(Answer: $0600 – which is the address of the machine language code for W1).

Here is a quick synopsis of the values stored in the IP, W, and hardware PC for the execution of EXAMPLE, given above. It might be a good idea to pause here, and try to run through the rest of the example on your own, to check your understanding (and the clarity of my explanation) of how NEXT functions.

Word-to-Execute	IP	W	IP-AT-EXIT	PC
W1	$E000	$A000	$E002	$0600
W2	$E002	$B000	$E004	$0700
W3	$E004	$C000	$E006	$0800
EXIT	$E006	$0900	$E008	$0880

As a final aid to understanding, here is an implementation of NEXT in hi-level Forth:

```
: NEXT  IP 2 +! // Get address of IP
        @       // Get value of IP (address of next word to
                execute)
        @       // Get that word's execution address
        W !     // And stuff into the current word pointer.
        2 IP +! // Move IP along to next word, for next time.
        W @     // Get the execution address from W.
        @       // Get the actual address of the code.
        PC !    // Force into hardware PC, so that it will
                execute.
        ;
```

Now that you understand NEXT (I hope), and the role of the Forth registers IP and W, you are in a good position to understand the rest of the Forth system.

EXECUTE – or how Forth launches programs

You might be wondering at this point exactly how an application gets launched in the first place. Since NEXT uses the IP, and assumes that the IP is pointing at a compiled execution address, how do words that you just type in from the terminal get executed? Obviously, words typed directly to the interpreter

from the terminal don't have an address which is valid for the IP. The answer is the Forth word EXECUTE, which takes an execution address as its argument. When you type a word to the interpreter that it can find in the dictionary, it pushes the execution address of the word onto the parameter stack, and calls EXECUTE. Execute first saves this execution address in W, and then forces the PC to the address stored in this execution address, just like the last part of NEXT. Here is EXECUTE in high level forth:

```
: EXECUTE  ( execution-address  ---  )
        W !      // save in W - then do last part of NEXT
        W @ @ // get the address of the code to execute
        PC !     // and execute it.
        ;
```

Note that EXECUTE does not call NEXT – it assumes the EXECUTEd word will be doing that.

At this point you are no doubt wondering how the IP gets initialized at all. It's not hard to understand, but let's put off a detailed discussion of it until we talk about the DOCOLON and EXIT routines, a little further on, but here is a brief hint: When EXECUTE executes your word, there is already a valid value in the IP – it is pointing somewhere inside INTERPRET. If the word you are executing is a colon definition, then the first thing it does is save the current value of the IP, and then changes it to point to itself. A CODE definition won't change the IP at all (unless the code you write is supposed to), and so the pointer to inside INTERPRET just hangs around until the code definition gets to its NEXT call, which causes the INTERPRET word to resume. This will all become clearer when you understand exactly how DOCOLON and EXIT work.

Incidentally, there is also an EXECUTE inside of the compiler loop – it's there to handle IMMEDIATE definitions – the ones that execute even when you are compiling. The logic here is the same as above. The only difference is that the IP will be pointing somewhere inside) , instead of inside INTERPRET.

How Forth Does Branching

In our discussion of NEXT, above, we only talked about sequential execution of words. What happens if we need to branch around words (as we do in conditionals like IF) or cause the same words to be executed repeatedly (as we do in DO LOOP or BEGIN UNTIL constructs)?

The answer is actually very simple – we just change the IP to point to the word we want to branch to, and then execute NEXT. If you followed the above discussion on NEXT, it should be obvious that this causes a complete diversion of the flow of control for the current word.

When a branch is compiled, two things are done: a special word that controls the branch is compiled, and the destination address of the branch is compiled. For example:

```
: CR'S  BEGIN  CR  AGAIN ;
```

This word will just print new lines, until a rude action is taken by the operator to stop it. Here is how the compiled word looks in memory.

```
                (Standard Forth Header goes here)
$A000  CR       ( execution address of CR)
$A002  BRANCH   ( execution address of BRANCH)
$A004  $A000    ( address to BRANCH to)
$A006  EXIT     ( execution address of EXIT)
```

When CR'S executes, NEXT executes CR, and then it executes BRANCH. BRANCH takes the address immediately following it in memory, in this case $A000, and stuffs it into the IP. BRANCH then JMP's to NEXT. Since the IP is once again pointing at CR, (having been changed by BRANCH), NEXT once again executes CR, and then BRANCH, which causes the IP to be changed, and so on, forever.

BRANCH is an example of an unconditional branching primitive – it always branches, no matter what. ?BRANCH is a conditional branching word – it will branch if the value on the top of the stack is FALSE – otherwise, no branch takes place. Here is an example of a word that would cause ?BRANCH to be compiled:

```
: CR? ( BOOLEAN -- ) IF  CR  THEN ;
```

CR? will obviously print a CR if the top of the stack is non–zero, otherwise, nothing happens. Here is how CR? would look in memory:

```
                (Standard Forth Header)
$A000  ?BRANCH  ( execution address of ?BRANCH)
$A002  $A006    ( destination address of branch )
$A004  CR       ( execution address of CR)
$A006  EXIT     ( execution address of EXIT)
```

In this case, the execution would execute ?BRANCH first, which tests the value of the top of the stack. Notice that two things can happen here, BOTH of which will change the IP:

1) If the top of the stack is FALSE, ?BRANCH will force the IP beyond the branch address, by adding two. This will obviously cause CR to be executed.

2) If the top of the stack is TRUE, ?BRANCH will act exactly like BRANCH, and stuff $A006 (the word immediately following ?BRANCH) into the IP, which will obviously just EXIT the definition.

In any branching word, one or the other of these two things will happen. All of the branching words are compiled in exactly this way, with the branching primitive first, and the destination address of the branch immediately following it in memory. The reason that there are more branching primitives in Blazin' Forth than just these two has more to do with entry and exit conditions that it does with the actual branching mechanism.

For example, IF–THEN, IF–ELSE–THEN, BEGIN–UNTIL, BEGIN–WHILE–REPEAT, BEGIN–AGAIN are all implemented with combinations of ?BRANCH and BRANCH, since all of these involve boolean testing of the top of the stack.

Things like DO–LOOP and ?DO–LOOP and DO–+LOOP, etc. have additional things to do, like move the loop parameters to the return stack, add or subtract the loop index, test the loop index, and clean up the return stack on the loop exit. But the actual mechanics of branching are exactly the same, only the entry/exit conditions differ from word to word. Among other advantages, it makes the compiler code much simpler, since there are fewer 'special cases' to check for.

Once again, as an aid to understanding, here are sample implementations of BRANCH and ?BRANCH in hi–level forth. As you read these, keep in mind that when BRANCH or ?BRANCH is executing, the IP will be pointing at the branch address – since it gets incremented before the execution of the next word by NEXT:

```
: BRANCH ( branch unconditionally STACK: -- )
      IP @    // Get the value of IP, ordinarily the address
              // of the code field of the next word to
              // execute. In this case, it is a branch address.
      IP @    // Get the value stored at the address – which
              // is the destination branch value.
      IP !    // Change the IP to the destination address.
      NEXT    // and execute.
      ;

: ?BRANCH ( conditional branch STACK: BOOLEAN -- )
      0= IF             // test top of stack – if FALSE ( equal
                        to zero)
         BRANCH // just execute BRANCH
      ELSE              // value was TRUE, don't branch.
         2 IP +!        // move IP over branch address, to
                        next word.
      THEN
      NEXT              // and execute.
      ;
```

How Forth Does Nesting – DOCOLON and EXIT

In the above examples there was never any question of remembering where we came from – the course of execution of the word was changed, and we never really cared to remember what called what. But what about having one colon definition calling another one? How does Forth remember where to come back to when it has finished the called definition?

This is not particularly difficult either. Once again, the IP and W, the current word pointer, play central roles. In what follows, remember that W points to the actual address of the word we want to execute, while the IP points to a memory location which contains the address of the word.

What happens is this: NEXT starts to execute a colon definition. All colon definitions have the same address stored at their execution address, which is the address of a machine language routine called DOCOLON or NEST. It is this routine that is responsible for saving the current execution environment.

DOCOLON first pushes the current value of the IP (which holds the address of the word we want to return to) onto the return stack. At this point, W will be holding the execution address of the new word to execute. We want to execute the body of this word, so DOCOLON now adds two to the value in W, which makes it point to the BODY of this definition, and stuffs it into the IP. DOCOLON now calls NEXT, which causes the new word to execute.

Eventually, NEXT will execute EXIT, which is the word compiled by ; . EXIT's job is to restore the previous execution environment, and it does this by very simply by pulling the top of the return stack, and stuffing it into the IP. It then calls NEXT, which causes the calling word to resume execution as though nothing had happened.

Here is an example:

```
: FOOBAR        CR ;

: COLON–CALL  FOOBAR ;
```

Compiled view of the above:

```
          (Header for FOOBAR)
$A000   DOCOLON ( Code field portion of header)
$A002   CR          ( Body )
$A004   EXIT

          (Header for COLON–CALL)
$B000   DOCOLON ( Code field portion of header)
$B002   FOOBAR    ( Body)
$B004   EXIT
```

And here is a simplified execution of COLON–CALL.

Word-to-Execute	IP	W	IP-AT-EXIT	RETURN-STACK
FOOBAR	$B002	$A000	$B004	xxxxx
DOCOLON	$B004	$A000	$A002	$B004
CR	$A002	CR's EA	$A004	$B004
EXIT	$A004	EXIT EA	$B004	xxxxx
EXIT (in CALL-COLON)	$B004	EXIT EA	-----	-----

(NOTE: EA stands for Execution Address.)

Once again, here is a sample implementation in higher level forth, of DOCOLON and EXIT:

```
: DOCOLON IP @  // get current value of IP
          >R    // Save it on return stack
          W @   // Get execution addr of current word.
          2 +   // Convert to Address of body.
          IP !  // Change IP
          NEXT // Execute new word
          ;

: EXIT    R<    // Get old IP (saved by DOCOLON)
          IP !  // Restore
          NEXT // Resume execution.
          ;
```

Since the most recent caller is always at the top of the return stack, the forth system can find its way through any number of levels of nesting, no matter how deep. There is no theoretical limit to the depth of nesting of forth words, although there is the practical limit of the size of the return stack.

So how deeply can one nest definitions in Blazin' Forth? Well, the obvious answer is around 123 levels, since there is an entire page of memory allocated for the return stack. It is equally obvious that certain actions can modify this, such as pushing literals to the return stack in your definitions, or using DO LOOPS, since DO LOOPS store the loop control information on the return stack.

Less obviously, you should note that CODE definitions do not cause the above nesting to occur. The majority of the primitives in Blazin' Forth are CODE definitions, and the desire to maximize the level of nesting was one of the design considerations that led to this decision.

In practice, I have never even approached the theoretical maximum level for nesting, much less had a crash that was traceable to return stack overflow, even when using highly recursive words.

Forth's DOES> construct

The implementation of the DOES> feature of Forth is usually one of the hardest for people to understand. The thing to remember when we get down to the actual details of the implementation is exactly how the current word pointer W works. When a word is executing, W will contain the execution address of that word. Stored at the address in W is the actual address of the code that is executing. In Forth, we would say that W @ is the execution address of the word, and W @ @ is the address of the code. Keeping this in mind will help you to understand what is going on.

First, a quick refresher on what DOES> does. DOES> is possibly the most unique feature of forth, since it allows you to extend the actual Forth compiler to compile new types of words. DOES> words are compiler words, and as such, they are used to create new words to execute. To help keep the discussion clear, lets call words which contain DOES> parent words, and words which are created by DOES> words, child words.

When a parent word executes, it creates a dictionary entry for the child. When the child executes, it leaves the address of its body on the parameter stack, and then executes the hi-level forth words after DOES> in the parent word. A common way to teach beginners about DOES> is to redefine one of the Forth primitives, such as CONSTANT, as a DOES> word. I'll do the same thing here, but I will also try to explain exactly how these words do their thing on an implementation level.

```
: CONSTANT  CREATE , DOES> @ ;
```

Here we have our CONSTANT definition. When CONSTANT (the parent) executes, it will create a dictionary entry with a standard header (that's the function of the CREATE in our definition). It then allocates two bytes of parameter space, and compiles the value on the top of the stack into the dictionary (that's the function of the ',' in our definition). The words following the DOES> don't do anything when CONSTANT executes – they execute when the child word (the word created by CONSTANT) executes. When the child executes, it will leave the address of its body on the stack, and then the words following the DOES> will be executed. In this example, there is only the @ – which will replace the address of the BODY with the value stored there, just like CONSTANT should, and EXIT, which will return us to wherever we came from. Thus:

```
10 CONSTANT TEN
```

creates a dictionary entry for the name TEN, and a 2 byte parameter field for the value 10, which CONSTANT also stuffs there. Executing

```
TEN
```

will first leave the address of TEN's body on the stack, and then the words following DOES> (in the parent word CONSTANT) will execute, which result in the value 10 being left.

Now for the implementation details. Here is how our definition of CONSTANT would look in the dictionary:

```
   (Preceded, as always, with the standard forth header)
$A000  DOCOL        ( code field portion of header)
$A002  CREATE       ( execution address )
$A004  ,            ( execution address )
$A006  (;CODE)      ( execution address )
$A008  JMP DODOES   ( actual machine lang. instructions)
$A00B  @            ( execution address)
$A00D  EXIT         ( execution address)
```

And here is how the definition of TEN would look:

```
( Standard dictionary header goes here)
$B000  $A008        ( code field portion of header)
$B002  10           ( value stored in parameter field)
```

Ok, here is how it all sorts out. Remember that DOES> is defined as an IMMEDIATE word, and so it executes when you are compiling. The mysterious portion of the CONSTANT definition, above – the (;CODE) and the JMP DODOES are written into the dictionary whenever DOES> executes.

(;CODE) is an unusual primitive. When it executes, it overwrites the current contents of the code field of the last word added to the dictionary with the address of the machine code which follows it in the definition currently executing. In our example above, it will cause all words created with CONSTANT to have a code field whose value is $A008 – the address of the JMP instruction in CONSTANT. This will obviously cause the JMP DODOES instruction to be executed each time a word created by CONSTANT is executed.

DODOES is the routine that does the actual magic. It must do three things:

1) It must save the current value of the IP (just like DOCOLON) so Forth knows how to get back to the caller.

2) It must push the address of the child's body to the stack.

3) It must execute the words following the JMP DODOES in the parent.

Using TEN as an example, DODOES must push the value $B002 to the parameter stack, and it must then cause the words starting at $A00B to be executed.

Here is how it's done in Blazin' Forth:

When TEN executes, it should be clear that the value stored in the current word pointer (W) is $B000, which is the execution address of TEN. The IP will be pointing somewhere important, so DODOES first saves it, which it does exactly like DOCOLON, by pushing it onto the return stack.

Once the IP has been safely tucked away, we have two tasks to perform. We must push the address of the parameter field of TEN to the stack, and we must then cause the hi-level forth words in the DOES> part of CONSTANT to execute. We can use the value of W to do both these things.

Remember that W, the current word pointer, is currently pointing at the execution address of TEN, and so contains $B000. So it is a simple matter to calculate the address of the body of 10 – we just add two to the current value of W (which gives us $B002), and push it to the parameter stack.

Now, the value stored at $B000 is $A008, which is the address of the JMP DODOES instruction in CONSTANT. We want to execute the hi-level forth words beyond this instruction – a piece of cake. We simply add 3 (the size of an absolute JMP instruction on the 6502) to the value stored at the execution address of the child word TEN, and stuff it into the IP. Once this has been done, all we need to do is call NEXT, which takes care of everything else, since we just pointed the IP at the proper spot.

Since we saved the previous value of the IP first off, when the EXIT at the end of the DOES> stuff is executed, we get returned to whatever called us.

Once again, here is an example of DODOES in hi-level forth:

```
: DODOES IP @ >R // Save current IP on return stack
         W @ 2 + // Leave address of parameter field on
                 stack
         W @ @   // Get address of JMP DODOES in-
                 struction
         3 +     // Add in size of JMP absolute instruc-
                 tion
         IP !    // Set as new execution address
         NEXT    // and execute it.
         ;
```

That wasn't so hard, was it?

LITERALS, CONSTANTS, and VARIABLES

In this last section, I talk about how Blazin' Forth handles compiled literals, and how the words defined by CONSTANT, VARIABLE and USER are implemented.

There are two kinds of literals recognized by Forth, numeric and string. Numeric literals are compiled automagically, by the compiler loop, while string literals are compiled by ." (usually).

Numeric literals first. As you probably remember, when you compile a definition, Forth attempts to look up each word in the definition in the dictionary. If it finds the word, then it compiles the execution address of the word into the dictionary (unless the word is defined IMMEDIATE, of course!). If the word is not found, then it attempts to convert the string of characters you just fed it into a number. If it succeeds, then it compiles a special primitive called (LIT) into the dictionary, and immediately past that, it places the value of your literal. (If it can't convert it to a number, then it issues the famous "NOT IN CURRENT SEARCH ORDER" message.) Here is an example:

```
: BIG  1000 ;
```

and here is how it looks in the dictionary:

```
            (standard forth header)
$A000  (LIT)  ( start of BIG's BODY)
$A002  1000 ( The value of your literal)
$A004  EXIT  ( EXIT – Tadah!)
```

(LIT)'s function is to place the value following it in memory on the top of the parameter stack, and to move the IP over the literal value, to the next valid forth word. It's pretty simple in practice, if you remember that if (LIT) is being executed, the IP must be pointing at the address of the literal, since it was incremented by NEXT. Here it is as an example FORTH definition:

```
: (LIT)  ( -- 16bit)
     IP @ @   // Get value of literal to stack.
     2 IP +!  // Move IP past literal value, to next valid
              word.
     NEXT     // and call NEXT
     ;
```

Blazin' Forth has a memory saving feature for values that will fit in one byte. For these values another word, called CLIT is compiled, instead of (LIT). It works very similarly to (LIT):

```
: CLIT ( --- byte-value)
     IP @ C@ // get the byte to the parameter stack
     1 IP +!  // move over byte literal to next valid word.
     NEXT     // and execute next
     ;
```

The case of string literals is very similar. ." is an immediate word which first compiles (."). It then searches the input stream for an ending ", and moves everything before this final quote into the dictionary, with a leading count byte, as is

normal for Forth. It also moves the pointers to the input stream past the string, so the interpreter won't try to evaluate it. Here is an example:

```
: GREETING  ." HELLO" ;
```

And here is how GREETING would look in memory:

```
        (Header)
$A000   (.")        ( primitive to print the following in-line
                    string)
$A002   5           ( the length of the string)
$A003   H E L L O   ( The chars are stored here, 1 per byte )
$A008   EXIT
```

The (.") primitive is one of the few low level words in Blazin' Forth that is actually written in Forth (i.e. it's a colon definition). Since (.") is a colon definition, this means that when (.") is called, DOCOLON will save the current value of the IP on the return stack. But, by a pretty stroke of fate, this will be exactly the address of the string following (."). To get a little more concrete about it:

When Greeting executes, the IP will eventually contain the value $A000. This will cause NEXT to execute (."), but NEXT will first, as always, bump the IP to $A002 (the start of the in-line string). When (.") executes, since it is a colon definition, DOCOLON will push $A002 (the current IP) to the return stack, and then enter the definition. So at entry, we have the address of the string on the return stack. All we have to do is retrieve the address, use COUNT and TYPE to display it, and adjust the return address on the stack before we exit. Once the return address has been adjusted and placed back on the stack, EXIT will return us to the word past the end of the in-line string.

Here is (."), just as it appears in the Blazin' Forth:

```
: (.") ( --- )
      R@        ( get string address from return stack)
      COUNT     ( get the count byte, adjust address )
      DUP 1 +   ( total length of string, including count byte)
      R> +      ( get address, move past end of string)
      >R        ( and restore, for EXIT)
      TYPE      ( the string)
      ;         ( and return, using adjusted address as re-
                turn)
```

So much for literals.

Constants and variables' (including USER variables) run-time action is determined by the routines pointed to by their code fields. There is no special primitives compiled, as there is with the literals.

Here is a short run-down of the actions of each: Constants place the value stored in their body on the parameter stack. Variables place the address of their body on the parameter stack. User variables place the address of the associated variable on the stack. The actual value stored in the parameter field is an offset from a base address.

Armed with your present knowledge of the IP and W, understanding these definitions should be a snap. They all work very much the same. We start with variable, since it's the simplest.

When the code field of a variable (or constant or USER) is executed, W will contain the execution address (the address of the code field) of the word in question. So it's easy: take the value in W, add two, and leave that value on the stack. Here is a hi-level definition of DOVARIABLE:

```
: DOVARIABLE ( -- address)
        W @     // Get the execution address of this
                variable
        2+      // Add two to get the body.
        NEXT    // and that's it!
        ;
```

Constants are very similar to variables – the only difference is the extra step required to retrieve the value in the constants body. Here is a hi-level definition of DOCONSTANT:

```
: DOCONSTANT ( -- value)
        W @ 2+ // As in variable - get the address of
                the body.
        @       // Get the value stored there.
        NEXT
        ;
```

USER variables are very similar to constants. The only addition here is that we add the base address of the user area to the value stored in the body of the user variable.

```
: DOUSER ( -- address )
        W @     // get execution address of this user
                variable
        2+ C@   // get offset - we only use byte offsets
                in Blazin' Forth.
        UP @    // get base address of user area
        +       // add to offset to get actual address of
                variable
        NEXT
        ;
```

Here is a question for those who want to test their general comprehension of the topics discussed here. Why can't we use the IP instead of W in the definitions of VARIABLE, CONSTANT, and USER ?

Answer: Aside from making the definition more complex, it would be impossible to retrieve the addresses of variables, or the values of constants, when we are typing their names directly into the interpreter from the terminal! Remember that the interpreter launches programs by stuffing the execution address of a word into W. In the following situation, there is no way to get from the address of the IP to the address of the parameter field:

```
VARIABLE BLETCH
BLETCH . XXXX
```

since the IP is still pointing somewhere inside INTERPRET. The only pointer that is valid to code such as DOVARIABLE in all cases is W.

Programmer's Aid For The Commodore 128

Joseph F. Caffrey
Larksville, PA

Find, Replace, and List Scrolling for 40 or 80 columns.

What more could one ask for than BASIC 7.0? When the initial thrill of playing with the new machine wears off, one notices that find and replace commands would be nice, together with a means to browse through the long listings you can create with all that available memory.

The program presented today fills these needs. The job of bidirectional list scrolling has been simplified compared to the C64, since Basic now contains a callable subroutine at $5123 which will list a single program line. Furthermore, we have a new system vector at $033C which can be used to intercept keystrokes before they are entered into the buffer. This vector is used by the program to take special action when a cursor up is entered with the cursor at the top left of the screen window, or a cursor down entered from the bottom left of the screen window. When this happens, instead of being stored in the buffer, the keystroke is suppressed, the screen is opened as a logical file, and the active screen area searched for a Basic line number at the left of a logical line, starting from the current cursor position. If a valid line number is found, a line is inserted on the screen and the next (or previous, if at the top of the window) line is listed to the screen. The cursor is then restored to the start of the new line, so holding down either cursor down or up causes scrolling through the listing.

If no valid line number is found after searching the whole active screen, the cursor move is simply executed.

Given the possibilities of various window shapes, 40 or 80 column output, and the inaccessibility of the 80 column chip memory in the memory map, cursor positioning by brute force would be a huge undertaking, so the kernal PRIMM subroutine, which prints text following its invocation up to a $00 byte, is used to send escape sequences to move the cursor about the screen. The code which determines which line to list next allows for the special cases of having found the final or initial line of the current program during the screen search by inserting two blank lines and "rolling over" to list the initial or final line as required.

After the initial installation, the list scrolling feature is transparent – it just happens with no action by the user – in fact, it even happens if you leave Basic for the monitor, which is a bit disconcerting, but doesn't appear to hurt anything!

The find and replace functions require creation of new Basic commands. The technique used here is patterned after Brian Munshaw's error wedge for the C64 (Transactor, Vol. 5, No. 6). The code traps a syntax error produced by a 'commercial at' (@) symbol. Rather than use literal phrases after the '@', I chose to follow it with a valid Basic 7.0 keyword. This allows the new command parser to check the single character following the '@' for dispatching purposes. I admit there are probably more elegant ways (there are a few more new vectors in page 3 of RAM) to do this, but I was anxious to get the code running.

Syntactically, the result is as follows. For "find", the command is:

```
@GET "text to be found"
```

This may be followed by a line number range, if desired (no comma, just the range immediately after the closing delimiter). Either single or double quotes may be used for the delimiters, with double quotes resulting in a search for the literal, nontokenized text enclosed therein.

The syntax for the replace function is similar:

```
@IF "text to change" THEN "replacement text"
```

Again, a line number range may follow, and single or double quotes may be used as delimiters.

The @GET command lists all lines containing the search text to the screen. @IF replaces all occurrences with the replacement string specified, and lists the changed line to the screen. Search and replace strings may be up to 32 characters long.

Because of the way the input line is tokenized, you can find REM's and DATA's, but changing them is difficult. If you need to specify a line number range after such a command, you can use the Commodore + Q graphic between the first and last line numbers to represent a tokenized minus (–) sign.

The replacement posed several problems. My first attempts involved temporarily resetting some system vectors, listing the changed line to the screen, poking a carriage return to the key buffer, letting Basic do the work of entry, then returning to my code. Unfortunately, Basic line entry involves calling a subroutine which resets the stack pointer, which confuses matters greatly. Replacing JSR's by JMP's and even defining a new page for the stack got me closer, but at that point I looked at the size to which the code had grown and decided to move the memory around myself, since it might be shorter and certainly faster than de-tokenizing and re-tokenizing a line which you had in tokenized form to begin with.

The resulting move routines do some tortuous arithmetic at their beginnings in order to set up pointers for a "let 'er rip" loop to effect the actual move. For search and replace strings of equal lengths the move routines are skipped, and if the replace string is longer than the search, checks are performed for creation of a line which is too long and (although I don't think it will happen soon) exhaustion of available memory.

The method of patching into Basic deserves some comment. The new capabilities of the 128 bring new responsibilities for ML programmers. Now in addition to all the old caveat's (don't hurt the stack, don't use some other routine's memory, don't mess up a register that some other routine is going to use), you must always be conscious of the current memory configuration the processor sees. As the Basic interpreter does its thing, you can only be sure that the first four pages of bank 0 RAM will always be visible to the processor. This is the reason for the patch code written to $03E4. The new error vector jumps to here, and we then make sure that the processor sees bank 0 at $14A0 before jumping there.

Also, the routines which read from bank 0 (READFM61 = $42EC, READFM70 = $42F1) leave the machine in a peculiar state. The ROM's are re-enabled, but the processor sees the character generator at $D000 instead of the I/O chips – this can wreak havoc with kernal routines, so Basic contains a number of six byte long code fragments which set up configuration 15 (which includes the I/O chips) and jump to kernal routines. We make use of two of these (BPRIMM = $9281, BCHROUT = $9269).

I've given it a lot of thought, but still can't see why you want the character generator in the processor's address space in the first place. Maybe the firmware was written by a committee?

As they stand, the @GET and @IF routines appear bug-free. Perhaps with syntax that ugly, one doesn't need bugs. I think I caught the last bug in the scroller a few days ago. It's really necessary to cancel quote mode after each list. Otherwise listing a line with an odd number of quotes on it results in a gentle crash, with the cursor bouncing back and forth between the first and second characters on the line. The interrupt is set at the start of the scroller to eliminate re-entry problems, which otherwise will insert blank lines randomly into your scrolling listing.

I have found two cases of aberrant behavior. When used with the 40 column screen, scrolling will present the first character of each line listed in reverse video. Never having mastered the vagaries of the software cursor on the 64, I leave this for others to fix. Besides, if it's a long program, you want to be on the 80 column screen anyway. Also, after sitting with my finger on the cursor down key for minutes at a time scrolling through a long listing, the routine has sometimes failed by listing an incorrect line number and proceeding to list there from. Fixes are welcome!

The routines are linked to Basic by a:

SYS DEC("1300")

Since a run/stop + restore sequence does not reset $0300, find and replace are there for good. However the $033C vector is reset. To allow scrolling to survive a restore sequence, the new error handler takes a few microseconds to rewrite that vector each time it is called, including, of course, immediately after the restore sequence.

C128 Progaid: BASIC Loader

```
FM 1000 rem save"0:progaid.ldr",8
GJ 1010 rem ** by joseph caffrey  larksville, pa
JB 1020 rem ** scroll, find and replace for the c128
PK 1030 for j=4864 to 5887: read x: poke j,x:
        ch=ch+x: next
HB 1040 if ch<>112895 then print"checksum
        error" : stop
GN 1050 print "sys(4864): rem to enable" : end
IJ 1060 :
JH 1070 data  76, 219,  22,  36, 127,  48,   9, 120
GA 1080 data  72,  41, 127, 201,  17, 240,   5, 104
AJ 1090 data  88,  76, 173, 198, 165,  46, 205,  17
OJ 1100 data  18, 208,   9, 166,  45, 232, 232, 236
FL 1110 data  16,  18, 240, 235, 104,  48, 103, 165
NI 1120 data 228, 197, 235, 208,   6, 165, 230, 197
KI 1130 data 236, 240,   4, 169,  17, 208, 217,  32
AG 1140 data 104,  20,  32,   4,  20,  32,  19,  20
JI 1150 data 144,  22, 165, 229, 197, 235, 208,   6
PL 1160 data  32, 113,  20,  76,  51,  19,  32, 129
EF 1170 data 146, 145,  27,  74,   0,  76,  61,  19
KK 1180 data  32, 113,  20, 169,  17,  32, 105, 146
DJ 1190 data  32, 100,  80, 144, 206,  32, 209,  22
JN 1200 data 134,  97, 133,  98, 160,   1,  32, 236
LK 1210 data  66, 208,  12,  32, 129, 146,  13,  13
FJ 1220 data   0, 166,  45, 165,  46, 208, 233,  32
IO 1230 data 130,  20, 169,   0, 133, 244,  32, 129
EF 1240 data 146,  27,  74,   0,  88,  96, 165, 229
LA 1250 data 197, 235, 208,   6, 165, 230, 197, 236
LE 1260 data 240,   4, 169, 145, 208, 151,  32, 104
CP 1270 data  20,  32,   4,  20,  32,  19,  20, 144
CJ 1280 data  20, 165, 228, 197, 235, 208,   6,  32
OA 1290 data 113,  20,  76, 154,  19, 169, 141,  32
DE 1300 data 105, 146,  76, 164,  19,  32, 113,  20
GP 1310 data  32, 100,  80, 144, 213,  32, 143,  20
NJ 1320 data 166,  97, 134, 250, 165,  98, 133, 251
KI 1330 data 197,  46, 208,  26, 228,  45, 208,  22
MF 1340 data  56, 173,  16,  18, 233,   2, 170, 173
MG 1350 data  17,  18, 233,   0, 134, 250, 133, 251
DD 1360 data  32, 143,  20,  32, 143,  20, 166,  45
IK 1370 data 165,  46, 134,  97, 133,  98,  32, 209
FM 1380 data  22, 197, 251, 208, 245, 228, 250, 208
OL 1390 data 241,  76, 127,  19, 169, 126, 162,   3
GL 1400 data  32, 186, 255,  32, 192, 255, 162, 126
FO 1410 data  76, 198, 255,  32,  92,  20, 176,  67
FJ 1420 data  41,  15, 133,  22, 169,   0, 133,  23
JK 1430 data  32,  92,  20, 144,   2,  24,  96,  41
MM 1440 data  15,  72,   6,  22,  38,  23, 176,  42
OE 1450 data 166,  22, 164,  23,   6,  22,  38,  23
HG 1460 data 176,  32,   6,  22,  38,  23, 176,  26
DA 1470 data 138, 101,  22, 133,  22, 152, 101,  23
HC 1480 data 133,  23, 176,  14, 104, 101,  22, 133
NM 1490 data  22, 169,   0, 101,  23, 133,  23, 144
HI 1500 data 199,  36, 104,  96,  32, 228, 255, 201
KM 1510 data  48, 144,   3, 201,  58,  96,  56,  96
GC 1520 data 165, 236, 133, 250, 165, 235, 133, 251
DM 1530 data  96, 169, 126,  32, 195, 255,  32, 204
CE 1540 data 255, 165, 250, 133, 236, 165, 251, 133
ND 1550 data 235,  96, 160,   2,  32, 236,  66, 170
MF 1560 data 200,  32, 236,  66,  76,  35,  81,  32
LJ 1570 data 129, 146,  27,  87,   0,  96,  72, 169
```

```
JD 1580 data   0, 141,   0, 255, 104,  76, 160,  20
KC 1590 data  32, 241,  22, 224,  11, 208,  37, 201
EO 1600 data  64, 208,  31,  32, 128,   3, 162,   1
IE 1610 data 221, 207,  20, 240,   5, 202,  16, 248
BM 1620 data  48,  16, 104, 104, 138,  10, 170, 189
OF 1630 data 210,  20,  72, 189, 209,  20,  72,  76
DJ 1640 data 128,   3, 162,  11,  76,  63,  77, 161
CG 1650 data 139, 212,  20, 212,  20, 134, 197, 240
GI 1660 data 241, 201,  34, 240,   4, 201,  39, 208
CI 1670 data 233, 133, 196, 162,   0,  32,  88,  21
KP 1680 data 132, 195, 165, 197, 240,  33, 162,  34
FL 1690 data  36, 127,  48, 216,  32, 132,  21, 201
CE 1700 data 167, 208, 207,  32, 128,   3, 197, 196
EL 1710 data 208, 200, 162,  32,  32,  88,  21, 132
CO 1720 data  32, 152,  56, 229, 195, 133,  94,  32
MN 1730 data 132,  21,  32, 251,  94,  76,  80,  21
DP 1740 data 160,   3, 162,   0, 200,  32, 236,  66
JP 1750 data 240,  39, 221,   0,  11, 208, 243, 232
KD 1760 data 228, 195, 208, 240, 165, 197, 240,   6
BE 1770 data  32, 195,  21,  32,  79,  79,  32, 130
OP 1780 data  20,  32, 152,  85,  32, 181,  75, 165
CJ 1790 data 197, 240,   6, 165,  34, 168, 136, 208
LD 1800 data 209,  32, 209,  22, 134,  97, 133,  98
DA 1810 data  32, 175,  22, 144, 195,  76, 134,   3
GK 1820 data  32, 128,   3, 160,   0, 142,   1, 255
LL 1830 data 177,  61, 142,   3, 255, 197, 196, 240
LJ 1840 data  18, 201,   0, 240,  19, 157,   0,  11
GH 1850 data 232, 200, 192,  31, 208, 231, 162,  23
OI 1860 data  76,  63,  77, 192,   0, 240,   1,  96
FM 1870 data 162,  11, 208, 244,  24, 152, 101,  61
ME 1880 data 133,  61, 165,  62, 105,   0, 133,  62
KP 1890 data  76, 128,   3, 141,   1, 255, 177,  89
NC 1900 data 145,  91, 200, 208, 249, 202, 240,   6
BH 1910 data 230,  90, 230,  92, 208, 240, 141,   3
FO 1920 data 255,  96, 142,   1, 255, 177,  89, 145
ON 1930 data  91, 136, 192, 255, 208, 247, 202, 240
PM 1940 data   6, 198,  90, 198,  92, 208, 238, 141
HJ 1950 data   3, 255,  96, 132,  34, 165,  94, 240
CB 1960 data  64,   8, 152,  24, 101,  97, 133,  89
PO 1970 data 165,  98, 105,   0, 133,  90, 173,  16
DP 1980 data  18,  56, 229,  89, 133,  91, 173,  17
EB 1990 data  18, 229,  90, 133,  92, 166,  92, 232
FA 2000 data  40,  16,  60, 165,  91,  73, 255, 168
LD 2010 data 200, 132,  91, 165,  89,  56, 229,  91
AA 2020 data 133,  89, 165,  90, 233,   0, 133,  90
KA 2030 data  32,  85,  22,  32, 147,  21,  32, 107
AO 2040 data  22, 165,  34,  56, 229, 195, 105,   0
MP 2050 data 168, 162,   0, 142,   1, 255, 189,  32
LH 2060 data  11, 145,  97, 200, 232, 228,  32, 208
HC 2070 data 245, 132,  34, 142,   3, 255,  96,  32
AE 2080 data 129,  22,  32, 159,  22, 164,  91, 136
FH 2090 data 165,  89,  24, 101,  91, 133,  89, 165
LC 2100 data  90, 101,  92, 133,  90, 132,  91,  56
NL 2110 data 165,  89, 229,  91, 133,  89, 165,  90
HL 2120 data 233,   0, 133,  90,  32,  85,  22,  32
KO 2130 data 170,  21,  76,   6,  22,  24, 165,  94
OJ 2140 data 101,  89, 133,  91, 165,  90, 133,  92
NO 2150 data 144,   2, 230,  92, 165,  94,  16,   2
FK 2160 data 198,  92,  96,  24, 165,  94, 109,  16
CN 2170 data  18, 141,  16,  18, 144,   3, 238,  17
EF 2180 data  18, 165,  94,  16,   3, 206,  17,  18
OK 2190 data  96,  24, 165,  94, 109,  16,  18, 172
```

```
OM 2200 data  17,  18, 144,   1, 200, 204,  19,  18
MF 2210 data 240,   3, 176,   6,  96, 205,  18,  18
FA 2220 data 144, 250, 162,  16,  76,  63,  77, 160
AA 2230 data   0,  32, 236,  66,  56, 229,  97, 101
OE 2240 data  94, 176,   1,  96,  76, 118,  21, 160
KD 2250 data   1,  32, 236,  66, 208,   3,  56, 176
AE 2260 data  23, 160,   3,  32, 236,  66, 197,  23
HA 2270 data 144,  14, 208,  12, 136,  32, 236,  66
GI 2280 data 197,  22, 144,   4, 240,   1,  36,  24
OA 2290 data  96, 160,   0,  32, 236,  66, 170, 200
CI 2300 data  76, 236,  66, 120, 162,  10, 189, 150
CF 2310 data  20, 157, 228,   3, 202,  16, 247, 169
NA 2320 data 228, 141,   0,   3, 169,   3, 141,   1
CB 2330 data   3, 120,  72, 169,   3, 141,  60,   3
IJ 2340 data 169,  19, 141,  61,   3, 104,  88,  96
```

PAL Source Listing

```
MK 1000 rem save"0:progaid.pal",8
AP 1010 rem c-128 scroll, find, replace
MP 1020 rem by joseph caffrey, larksville, pa
LH 1030 open 8,8,1,"0:progaid.obj"
BO 1040 sys700
ND 1050 .opt o8
KJ 1060 ;
MM 1070 ;constants
OK 1080 ;
BO 1090 bitins    =  $24          ;opcode for bit zp instruction
IL 1100 crsdwn    =  17           ;ascii characters
KI 1110 crsup     =  crsdwn + 128
HA 1120 escape    =  27
JJ 1130 wdgchr    =  "@"
GG 1140 shiftrtn  =  $0d + 128
BD 1150 synerr    =  11           ;basic error #'s
OE 1160 nomemory  =  16
EN 1170 dironly   =  34
AH 1180 lngstrng  =  23
CD 1190 thentoken=   $a7          ;basic token for "then"
GC 1200 ;
LF 1210 ;zero page locations used
KD 1220 ;
EN 1230 linenum   =  $16          ;zp storage for line #
EH 1240 replen    =  $20          ;temp string area
CE 1250 pickup    =  $22          ;"
MF 1260 sob       =  $2d          ;holds start addr of workspace
HN 1270 txtptr    =  $3d          ;in chrget routine
PI 1280 source    =  $59
OH 1290 dest      =  source + 2
EB 1300 temp      =  dest + 2
PD 1310 shift     =  temp + 1
DO 1320 ;above zp loc's in fp math area
LN 1330 ;will be used for mem moves
FB 1340 linkpntr  =  $61          ;for searching thru basic prgrm
LL 1350 runmode   =  $7f          ;0 if direct, 128 if running
KD 1360 srchlen   =  $c3          ; used by tape load
BA 1370 delimit   =  $c4          ;"
BI 1380 flag      =  $c5          ;tape load/save
PD 1390 botline   =  $e4          ;screen editor variabes
JK 1400 topline   =  $e5
GC 1410 leftedge  =  $e6
CH 1420 riteedge  =  $e7
AL 1430 xpos      =  $ec
MI 1440 ypos      =  $eb          ;cursor position
BE 1450 qtsw      =  $f4          ;quote swtch used by screen ed
HB 1460 ;above val's relative to top left
KN 1470 ;of physical screen- not window!
PG 1480 xsave     =  $fa
GO 1490 ysave     =  $fb          ;unused by system
CF 1500 ;
KD 1510 ;system vectors and pointers
GG 1520 ;
ON 1530 basbuf    =  $0200        ;basic input buffer
LO 1540 ierror    =  $0300
LE 1550 ikeystr   =  $033c        ;vector for store key routine
CC 1560 chrget    =  $0380
HP 1570 chrgot    =  $0386
JK 1580 patch     =  $03e4        ;free space in common ram
JL 1590 eob       =  $1210        ;holds nd addr of program
LN 1600 baslim    =  $1212        ;end of workspace
KA 1610 casbuf    =  $0b00        ;c128 cassette buffer
ED 1620 srchbuf   =  casbuf       ;use cassette buffe
OF 1630 repbuf    =  casbuf + $20
ON 1640 ;
MF 1650 ;system rom locations
CP 1660 ;
EN 1670 keystr    =  $c6ad        ;old key store routine
IL 1680 mmu       =  $ff00        ;memory management unit
DM 1690 setlfs    =  $ffba
DO 1700 open      =  $ffc0
LP 1710 close     =  $ffc3
AE 1720 chkin     =  $ffc6
OK 1730 clrchn    =  $ffcc
```

```
CL 1740 getin    =   $ffe4
ME 1750 ;
MC 1760 ;basic rom routines
AG 1770 ;
FB 1780 readfm61 =   $42ec      ;lda (61),y fm bank 0
HF 1790 readfm70 =   $42f1      ;lda ($70),y fm bank 0 ram
DF 1800 basstpky =   $4bb5      ;check stop key-break if pressed
JD 1810 errout   =   $4d3f      ;normal basic error handler
AD 1820 relink   =   $4f4f      ;relink basic wrkspace
NA 1830 huntline =   $5064
LB 1840 list1    =   $5123      ;list 1 line fm workspace
OM 1850 basiccr  =   $5598      ;print a carriage return
IJ 1860 linerng  =   $5efb      ;calculate line range for find/replace
MF 1870 bprimm   =   $9281      ;get in bank 15-print following text
OB 1880 bchrout  =   $9269      ;restore bank 15 & jmp chrout
HE 1890 *        =   $1300
LB 1900          jmp hookup
MO 1910 ;
BO 1920 newkystr =   *          ;check for cursor moves off screen
AA 1930 ;
JK 1940          bit runmode
CM 1950          bmi bailout    ;if running program
BE 1960          sei
EL 1970          pha            ;turn off kybrd rupt
GJ 1980          and #$7f       ;to eliminate reentrancy
KC 1990          cmp #crsdwn    ;is it either up or down
OI 2000          beq onward     ;we may have to do something
CG 2010          pla
OD 2020 bailout  cli
NM 2030          jmp keystr
JN 2040 onward   lda sob+1
BD 2050          cmp eob+1
OH 2060          bne onward1
BJ 2070          ldx sob
MP 2080          inx
GA 2090          inx
DL 2100          cpx eob
PD 2110          beq bailout-1  ;if no text
DK 2120 onward1  pla            ;which key is it
BM 2130          bmi upward     ;if it's cursor up
NJ 2140 downward =   *          ;it's a cursor down!
OI 2150          lda botline
PF 2160          cmp ypos       ;are we on bottom line-eh
HN 2170          bne downout    ;if not
KJ 2180          lda leftedge
FA 2190          cmp xpos
EF 2200          beq thisisit
EL 2210 downout  lda #crsdwn
PC 2220          bne bailout    ;just print cursor down
MI 2230 thisisit =   *          ;cursor down fm bottom left
HD 2240          jsr savecrsr   ;store cursor position
ID 2250          jsr opnscreen  ;turn on screen for input
FC 2260 chkline  jsr huntnmbr   ;look for a line number on screen
HD 2270          bcc foundit    ;if number found
GN 2280          lda topline    ;else check if at
JG 2290          cmp ypos       ;top of window and
PI 2300          bne moveitup   ;look more if not
BI 2310          jsr putback
PN 2320          jmp downout
AE 2330 moveitup =   *          ;move cursor up to next line
EI 2340          jsr bprimm     ;send text
BB 2350 .byte crsup,escape,$4a,0 ;cursor up,escape,"j"
LI 2360          jmp chkline
FK 2370 foundit  =   *          ;line number found
AL 2380          jsr putback    ;restore cursor & i/o channel
FI 2390          lda #crsdwn
FD 2400          jsr bchrout    ;print a cursor down
LF 2410          jsr huntline   ;find line within basic workspace
NC 2420          bcc downout    ;if # read is not a line #
LK 2430 ;last line listed has been located within basic wrkspace
LN 2440 ;now get link addr of next line
EM 2450          jsr rdnwlink   ;read new link bytes
GL 2460 setitup  stx linkpntr
LK 2470          sta linkpntr+1 ;point at next line
NP 2480          ldy #1
HL 2490          jsr readfm61   ;check for zero
LB 2500 ;which means line on screen was last line-
PD 2510 ;of prgrm so must list first
BC 2520          bne setup1     ;if not at end of prgrm
KJ 2530          jsr bprimm
CA 2540 .byt $0d,$0d,00         ;print two blank lines
BH 2550          ldx sob
HP 2560          lda sob+1      ;point at first line
JK 2570          bne setitup    ;always (basic never starts on zero page)
JM 2580 setup1   jsr listit
JA 2590          lda #0
HG 2600          sta qtsw       ;cancel quote mode just in case
JL 2610          jsr bprimm     ;restore cursor to left of line
PK 2620 .byt escape,$4a,0       ;by printing esc-j
EN 2630          cli
PH 2640          rts            ;and return
AN 2650 ;
CI 2660 upward   =   *          ;for cursor up at top
KP 2670          lda topline
AP 2680          cmp ypos
OI 2690          bne upout
CK 2700          lda leftedge
NA 2710          cmp xpos
NA 2720          beq itsanup
GP 2730 upout    lda #crsup
HI 2740          bne downout+2  ;long branch to bailout
ED 2750 ;
IM 2760 itsanup  =   *
FF 2770          jsr savecrsr
OO 2780          jsr opnscreen
DN 2790 chklinup jsr huntnmbr
NN 2800          bcc foundup    ;we found a line #
CC 2810          lda botline
GI 2820          cmp ypos       ;are we at bottom of screen
MA 2830          bne movitdwn   ;look lower if not
IB 2840          jsr putback    ;fix cursor & i/o
JB 2850          jmp upout      ;& print the cursor up
CK 2860 ;
PD 2870 movitdwn =   *          ;move cursor down screen
BK 2880          lda #shiftrtn
NA 2890          jsr bchrout    ;print shifted return
MG 2900          jmp chklinup   ;go back and do it agn
BD 2910 foundup  =   *          ;we got a line #
OE 2920          jsr putback    ;fix i/o & cursor
CF 2930          jsr huntline   ;see if it really exists
DD 2940          bcc upout      ;if it's false
HA 2950          jsr scroldwn   ;make blank line at top
AE 2960 ;now hunt from beginning of basic for line to be listed
AD 2970 oldlnklo =   xsave
FA 2980 oldlnkhi =   ysave      ;zp storage for old link addr (line read fm screen)
GJ 2990          ldx linkpntr
BK 3000          stx oldlnklo
KF 3010          lda linkpntr+1
ME 3020          sta oldlnkhi   ;save link addr of top screen line
GH 3030          cmp sob+1      ;was screen line first in prgrm
CH 3040          bne init
HH 3050          cpx sob
HM 3060          bne init       ;if not at start
PH 3070          sec
JB 3080          lda eob
LI 3090          sbc #02        ;else aim at
HO 3100          tax
AB 3110          lda eob+1
IN 3120          sbc #0         ;value stored in link of
DC 3130          stx oldlnklo
DH 3140          sta oldlnkhi   ;last program line
PD 3150          jsr scroldwn
HG 3160          jsr scroldwn   ;print 2 blank lines
BN 3170 init     ldx sob
EG 3180          lda sob+1
MH 3190 rstlink  stx linkpntr
GF 3200          sta linkpntr+1
EN 3210 ;aim linkpntr at appropriate address
NF 3220          jsr rdnwlink   ;get link of next line
OG 3230          cmp oldlnkhi   ;do we have a match
FB 3240          bne rstlink    ;keep lookin' if not
GD 3250          cpx oldlnklo   ;is it really a match
HG 3260          bne rstlink    ;look more if not
CN 3270          jmp setup1     ;list line with linkpntr intact
GE 3280 ;
PM 3290 opnscreen=   *          ;set screen as input device
MJ 3300          lda #126       ;file #
GF 3310          ldx #3         ;dev #
CM 3320          jsr setlfs
LH 3330          jsr open       ;open126,3
DM 3340          ldx #126
CH 3350          jmp chkin      ;make screen input device & rts
GJ 3360 ;
MK 3370 huntnmbr =   *          ;hunt for line number on screen
CF 3380 ;return carry clear if digit was read
PJ 3390          jsr get1       ;read 1 char fm screen
LG 3400          bcs doneline   ;if no digit found
IE 3410 ;here valid digit 0-9 was found
BG 3420          and #$0f       ;convert to binary 0-9
HM 3430          sta linenum
LF 3440          lda #0
JE 3450          sta linenum+1  ;set up for calculation
EO 3460 read1more jsr get1
FD 3470          bcc addin      ;read through all following digits
OA 3480          clc
JA 3490          rts            ;after last digit found
ON 3500 addin    and #$0f
DK 3510          pha            ;convert and save digit
GC 3520          asl linenum
PB 3530          rol linenum+1
AK 3540          bcs badnmbr    ;ln x 2
NF 3550          ldx linenum
FK 3560          ldy linenum+1  ;save x2 value
IF 3570          asl linenum
BF 3580          rol linenum+1
OC 3590          bcs badnmbr
GH 3600          asl linenum
PG 3610          rol linenum+1
JL 3620          bcs badnmbr    ;now = x8
CO 3630          txa
NE 3640          adc linenum
DG 3650          sta linenum    ;carry is clear
DA 3660          tya
EE 3670          adc linenum+1
KJ 3680          sta linenum+1
AG 3690          bcs badnmbr    ;now x10
MP 3700          pla
DJ 3710          adc linenum
JK 3720          sta linenum    ;carry is clear
NH 3730          lda #0
KI 3740          adc linenum+1
AO 3750          sta linenum+1
ON 3760          bcc read1more
AO 3770 .byt bitins             ;mask out pla on fallthrough
ML 3780 badnmbr  pla
MO 3790 doneline rts
OE 3800 ;
LG 3810 get1     =   *          ;read 1 fm screen-check if digit
```

```
ON 3820           jsr   getin
CC 3830           cmp   #"0"
EK 3840           bcc   less0
GC 3850           cmp   #":"          ;1 more than "9"
MG 3860           rts                 ;w/cc for "0-9"
EN 3870 less0     sec
EB 3880           rts
CH 3890 savecrsr  lda   xpos
CG 3900           sta   xsave
JH 3910           lda   ypos
HH 3920           sta   ysave
GE 3930           rts
GG 3940 putback   lda   #126
JH 3950           jsr   close         ;close screen file
LL 3960           jsr   clrchn        ;& restore io channels
KG 3970           lda   xsave
MP 3980           sta   xpos
PH 3990           lda   ysave
JN 4000           sta   ypos          ;restore cursor position
GJ 4010           rts
KC 4020 ;
GC 4030 listit    =     *             ;read line # into a&x
AD 4040           ldy   #2            ;point at line# lo byte
NC 4050           jsr   readfm61      ;read it
FD 4060           tax                 ;save it
BC 4070           iny                 ;point at line# hi byte
LE 4080           jsr   readfm61      ;read it
NL 4090           jmp   list1         ;let basic list line
KH 4100 ;
FG 4110 scroldwn  jsr   bprimm        ;print escape "w"
BB 4120 .byt escape,"w",0             ;to scroll down
OA 4130           rts
CK 4140 ;
ED 4150 ;error wedge routine for c128
GL 4160 ;
AF 4170 ptchcode  =     *             ;to be written to page 3
AN 4180           pha
JN 4190           lda   #00
ML 4200           sta   mmu           ;get in bank 15
KP 4210           pla
BJ 4220           jmp   start
MP 4230 ;
AH 4240 start     =     *             ;arrive here on error
HM 4250 patchlen  =     start-ptchcode
LC 4260           jsr   rstsysvc      ;restore vector for scroll routine
PG 4270           cpx   #synerr
CB 4280           bne   wrong+2
BP 4290           cmp   #wdgchr
IB 4300           bne   wrong
ME 4310 ;
IP 4320 ourerr    jsr   chrget        ;parse added commands and dispatch
IK 4330           ldx   #numcmds-1
GA 4340 chktable  cmp   cmdtable,x
II 4350           beq   gotit         ;if command found
PL 4360           dex
AC 4370           bpl   chktable      ;keep checking
PB 4380           bmi   wrong         ;report syntax error otherwise
MJ 4390 ;
LC 4400 gotit     =     *             ;here we execute command
CM 4410           pla
OI 4420           pla                 ;remove one return address
CA 4430           txa
EE 4440           asl   a
MJ 4450           tax                 ;multiply x by 2
OK 4460           lda   jmptbl+1,x    ;hi byte
CP 4470           pha
CG 4480           lda   jmptbl,x      ;lo byte
LG 4490           pha                 ;set up return addr
CL 4500           jmp   chrget        ;get next chr & rts to new routine
EB 4510 ;
GI 4520 wrong     ldx   #synerr
FI 4530           jmp   errout
CD 4540 ;
NB 4550 cmdtable  =     *
FE 4560 .byte $a1                     ;"get" token for find
EL 4570 .byte $8b                     ;"if" token for replace
KF 4580 ;
JN 4590 jmptbl    =     *
PL 4600 numcmds=  jmptbl-cmdtable
IH 4610 ;
MF 4620 .word find-1 ,replace-1
MI 4630 ;
OF 4640 ;128 find&replace
EO 4650 find      stx   flag          ;=0 for find, 2 for
HC 4660 replace   =     find
FN 4670           beq   wrong         ;if no params
NF 4680           cmp   #$22          ;look for double quote
OO 4690           beq   parsit
EE 4700           cmp   #"'"          ;or single quote
GO 4710           bne   wrong         ;only two above allowed
BH 4720 parsit    sta   delimit       ;save delimiter
KC 4730           ldx   #0            ;aim at search buffer
AM 4740           jsr   getstrng
HK 4750           sty   srchlen       ;length of search string-(3d) pointing at delimiter
NL 4760           lda   flag
OP 4770           beq   adjust        ;if not replacing
CC 4780 ;
LD 4790           ldx   #dironly      ;for possible error
FN 4800           bit   runmode
LM 4810           bmi   wrong+2       ;check for direct mode if replacing
AP 4820           jsr   resetptr      ;fix textptr
NP 4830           cmp   #thentoken    ;must have token for then-
PE 4840           bne   wrong         ;else syntax error
EB 4850           jsr   chrget
EJ 4860           cmp   delimit       ;then must have delimiter
EA 4870           bne   wrong         ;else error
GI 4880 ;
JB 4890           ldx   #$20          ;point at repbuf
NF 4900           jsr   getstrng      ;read replace string
AP 4910           sty   replen        ;replacement string length
LK 4920           tya                 ;replen to a
DM 4930           sec
FP 4940           sbc   srchlen       ;find shift
AN 4950           sta   shift
IF 4960 adjust    jsr   resetptr      ;point chrget at delimiter
AO 4970 ;
JO 4980 getlnrng  jsr   linerng       ;setup line range in (61) and (16)
MD 4990 ; (61) holds point in basic workspace where search should start
GC 5000 ;(16) holds highest line # which should be searched-now check if in range
JH 5010           jmp   chkit
CB 5020 ;
LF 5030 search1   =     *             ;search thru program line for the search string
MD 5040           ldy   #3            ;to start off point at hi byte line #
CJ 5050 search11  ldx   #0            ;point to start of search buffer
PA 5060 search12  iny                 ;point to next byte
DB 5070           jsr   readfm61      ;get byte fm wrkspace
PJ 5080           beq   donextln      ;if at nd of line
DG 5090           cmp   srchbuf,x     ;check for match
AI 5100           bne   search11
CN 5110           inx
FP 5120           cpx   srchlen       ;do we have good match
DE 5130           bne   search12      ;keep searching if not-but leave x pointing into buffer
KI 5140 ;
MA 5150           lda   flag          ;what are we doing-eh
OB 5160           beq   list1ln       ;if search and no replace
JK 5170           jsr   insert        ;put line in basic wkspc
GG 5180           jsr   relink        ;and fix up links
IL 5190 list1ln   =     *             ;list found or changed line
OE 5200           jsr   listit
PM 5210           jsr   basiccr
AE 5220           jsr   basstpky      ;provide for bailout via stopkey
MJ 5230           lda   flag          ;test for find/replace
MG 5240           beq   donextln      ;if finding
CA 5250 recheck   =     *             ;else go back and hunt for further matches
GG 5260           lda   pickup        ;points one beyond replace string
FG 5270           tay
CH 5280           dey                 ;set up for further search-dey 'cuz search will iny
BM 5290           bne   search11      ;go back and do again!
JC 5300 donextln  =     *
CA 5310           jsr   rdnwlink      ;adjust (61)
OO 5320           stx   linkpntr
BJ 5330           sta   linkpntr+1    ;hi byte next link addr
DF 5340 chkit     jsr   chkhiln
NI 5350           bcc   search1
CK 5360           jmp   chrgot        ;and thence to basic
AH 5370 ;
AG 5380 getstrng  =     *             ;read a string from chrget into buffer
OD 5390           jsr   chrget        ;advance pointer
DG 5400           ldy   #0
LB 5410 getstr1   stx   mmu+1         ;switch in bank 0
MC 5420           lda   (txtptr),y
JE 5430           stx   mmu+3         ;switch in rom's
JB 5440           cmp   delimit       ;at end of string
KJ 5450           beq   doneget
EI 5460           cmp   #0
GH 5470           beq   wrong2        ;check for line end before delimiter
EL 5480           sta   srchbuf,x
OE 5490           inx
MF 5500           iny
CN 5510           cpy   #31           ;maximum string length
DJ 5520           bne   getstr1       ;or fall thru to error
JN 5530 toolong   ldx   #lngstrng
EF 5540 wrong1    jmp   errout
NA 5550 doneget   cpy   #0            ;check for zero length
LE 5560           beq   wrong2
FK 5570           rts                 ;with string length in y
CE 5580 ;
OG 5590 wrong2    ldx   #synerr
NF 5600           bne   wrong1
AG 5610 ;
IK 5620 resetptr  clc
FL 5630           tya
KJ 5640           adc   txtptr
GN 5650           sta   txtptr        ;fix lo byte
OH 5660           lda   txtptr+1
DA 5670           adc   #0
LA 5680           sta   txtptr+1      ;
AH 5690           jmp   chrget        ;rts thence with next char
KL 5700 ;
JN 5710 movedown  =     *             ;move memory downward
OM 5720 ;
CK 5730           sta   mmu+1         ;into bank0
LH 5740 move1dwn  lda   (source),y    ;on entry
OF 5750           sta   (dest),y      ;x=#pages to be moved +1
FC 5760           iny                 ;y=256-#excess bytes
CK 5770           bne   move1dwn      ;source=start of block-y
FJ 5780           dex                 ;dest=source-shift
BG 5790           beq   donedown
HP 5800           inc   source+1
NF 5810           inc   dest+1
FD 5820           bne   move1dwn
HE 5830 donedown  sta   mmu+3         ;turn on rom's
ML 5840           rts
AF 5850 ;
OH 5860 moveup    =     *             ;move mem upward
EG 5870 ;
EJ 5880           stx   mmu+1         ;into bank0
DJ 5890 move1up   lda   (source),y    ;when called
```

```
DE 5900          sta  (dest),y     ;y=#excess bytes-1
JH 5910          dey              ;x=#pages to move+1
FM 5920          cpy  #$ff        ;source=end of block+1-#excess bytes=end-y
HD 5930          bne  move1up     ;dest=source+shift
LO 5940          dex
OK 5950          beq  doneup
CH 5960          dec  source+1
IN 5970          dec  dest+1
KK 5980          bne  move1up
HO 5990 doneup   sta  mmu+3       ;back to orig bank
MF 6000          rts
AP 6010 ;
GD 6020 insert   =    *           ;insert replacement string in program
EA 6030 ;
KJ 6040          sty  pickup      ;points to final byte search string
OC 6050          lda  shift       ;check if move needed
IF 6060          beq  stuffit     ;go ahead if not
EI 6070          php              ; save for later
FK 6080          tya              ;end of search string
AE 6090          clc
CF 6100          adc  linkpntr
JI 6110          sta  source
AI 6120          lda  linkpntr+1
PM 6130          adc  #0
DH 6140          sta  source+1    ;set source=start of block
HB 6150          lda  eob
BJ 6160          sec
HJ 6170          sbc  source
BO 6180          sta  dest
IB 6190          lda  eob+1
HH 6200          sbc  source+1
EN 6210          sta  dest+1      ;set dest=blocksize
DD 6220          ldx  dest+1
PM 6230          inx              ;x-reg now set with #pages+1
EC 6240          plp
CC 6250          bpl  slewup      ;if moving up
GO 6260 slewdown=    *           ;else move down
NP 6270          lda  dest
AD 6280          eor  #$ff
BG 6290          tay
HM 6300          iny              ;y-reg now set
NL 6310          sty  dest        ;temporary
NB 6320          lda  source
LD 6330          sec
DF 6340          sbc  dest
JH 6350          sta  source
HA 6360          lda  source+1
NN 6370          sbc  #0
NI 6380          sta  source+1    ;adjust source
AH 6390          jsr  fixdest     ;common to up and down
FP 6400          jsr  movedown
AJ 6410          jsr  fixeob      ;common to up and down
KI 6420 ;
OJ 6430 stuffit  =    *           ;insert replace string
PO 6440          lda  pickup      ;final byte search string
DL 6450          sec
HF 6460          sbc  srchlen
JP 6470          adc  #0          ;point to first byte search string
FG 6480          tay              ;point y at replacement
BK 6490          ldx  #0
AA 6500          stx  mmu+1       ;into bank 0
AJ 6510 stuff1   lda  repbuf,x
CK 6520          sta  (linkpntr),y
CG 6530          iny
IG 6540          inx
EE 6550          cpx  replen
MN 6560          bne  stuff1
PI 6570          sty  pickup      ;aimed one beyond rep string
DI 6580          stx  mmu+3       ;restore rom's
OG 6590          rts              ;replacement complete
OD 6600 ;
ND 6610 slewup   =    *           ;move mem upward
II 6620          jsr  roomatop    ;check if mem available
FF 6630          jsr  longline    ;check if line will be too long
PM 6640          ldy  dest
IA 6650          dey              ;y set for move
BH 6660          lda  source
EI 6670          clc
JI 6680          adc  dest
NM 6690          sta  source
LF 6700          lda  source+1
DL 6710          adc  dest+1
NM 6720          sta  source+1    ;source pnting to end of block
BG 6730          sty  dest        ;temporary
FN 6740          sec
LM 6750          lda  source
HP 6760          sbc  dest
NB 6770          sta  source
LK 6780          lda  source+1
BI 6790          sbc  #0
NP 6800          sta  source+1
GE 6810          jsr  fixdest
PG 6820          jsr  moveup
PL 6830          jmp  stuffit-3   ;fix eob and rewrite line
OC 6840 ;
HG 6850 fixdest  =    *           ;adjust dest pointer relative to source
CE 6860          clc
CB 6870          lda  shift
PD 6880          adc  source
MP 6890          sta  dest        ;low byte done
DC 6900          lda  source+1
NB 6910          sta  dest+1      ;hi byte half done
FA 6920          bcc  checkdir
OB 6930          inc  dest+1      ;add 1 if carry set
FO 6940 checkdir lda  shift
GP 6950          bpl  donedest
CD 6960          dec  dest+1      ;sub 1 if shift<0
DJ 6970 donedestrts
KL 6980 ;
CD 6990 fixeob   =    *
OM 7000          clc
OJ 7010          lda  shift
PG 7020          adc  eob
DK 7030          sta  eob         ;low byte done
EB 7040          bcc  checkdr1
DI 7050          inc  eob+1       ;add 1 if cs
AG 7060 checkdr1 lda  shift
OF 7070          bpl  doneeob
NI 7080          dec  eob+1
MO 7090 doneeob  rts
CD 7100 ;
GE 7110 roomatop=    *           ;check for room in mem
GE 7120          clc
GB 7130          lda  shift
HJ 7140          adc  eob         ;<new eob in a
EF 7150          ldy  eob+1       ;<hi byte
KF 7160          bcc  roomenuf
PG 7170          iny              ;if page boundary will be crossed
AK 7180 roomenuf cpy  baslim+1
IM 7190          beq  litlroom
NN 7200          bcs  noroom
FA 7210 roomok   rts
KN 7220 litlroom cmp  baslim
NI 7230          bcc  roomok
IK 7240 noroom   ldx  #nomemory
FC 7250          jmp  errout
CN 7260 ;
IA 7270 longline =    *           ;test if changed line will be too long
LL 7280          ldy  #0
DP 7290          jsr  readfm61    ;get <link to next line
FA 7300          sec
AJ 7310          sbc  linkpntr    ;find length of current line
MP 7320          adc  shift       ;get new length
MO 7330          bcs  wrong3      ;if line > 1 page
IJ 7340          rts
BD 7350 wrong3   jmp  toolong
MN 7360 chkhiln  ldy  #1
CG 7370          jsr  readfm61    ;get line link hi byte
KE 7380 ;
FM 7390          bne  comphi      ;nonzero hi link means not at end of program
JG 7400          sec
FP 7410          bcs  done        ;else go back carry set
JL 7420 comphi   ldy  #3
JI 7430          jsr  readfm61    ;get line # hi byte
HO 7440          cmp  $17         ;are  we finished
MP 7450          bcc  done        ;if lower than 17 read thru text-not done!
JL 7460          bne  done        ;if higher than 17 we are finished
CJ 7470 ;if hi byte ln# =(17) then check low byte
DP 7480          dey
FN 7490          jsr  readfm61    ;get line # lo byte
BC 7500          cmp  $16
IE 7510          bcc  done        ;if lower than 16 carry clear
HI 7520          beq  domore      ;if equal to 16 otherwise fall thru with carry set to
DA 7530 .byt bitins               ;bit zp  intruction
HE 7540 domore   clc              ;to indicate more to do
EI 7550 done     rts              ;return with carry clear if in range- carry set if beyond
AB 7560 ;(61),0 still holds currentline link low addr
MC 7570 ;so reaim (61)
CB 7580 ;
GO 7590 rdnwlink ldy  #0
II 7600          jsr  readfm61
CM 7610          tax              ;low byte of next link addr
EK 7620          iny
ML 7630          jmp  readfm61    ;link into x,a
OE 7640 ;
DJ 7650 hookup   =    *           ;link into operating system
CG 7660 ;
PI 7670          sei
GL 7680          ldx  #patchlen
OP 7690 patchlup lda  ptchcode,x
MD 7700          sta  patch,x
FN 7710          dex
FM 7720          bpl  patchlup
JM 7730          lda  #<patch
DK 7740          sta  ierror
JN 7750          lda  #>patch
OJ 7760          sta  ierror+1
LI 7770 rstsysvc =    *           ;revector system keypress store vector
NP 7780          sei
KO 7790          pha
OL 7800          lda  #<newkystr
LC 7810          sta  ikeystr
OM 7820          lda  #>newkystr
IB 7830          sta  ikeystr+1
IC 7840          pla
ID 7850          cli
AK 7860          rts
```

Commodore 64
23K 'RAM Disk'

Anthony Bertram
Toronto, Ontario

Make use of all that unused RAM in the 64!

Here's a utility that makes use of all the unused RAM in the C64, and is especially useful to those who are wishing for a second (or first!) disk drive. It can hold a single program as large as 93 blocks (23,808 bytes) and will save or load in either direct or program modes at blinding speed.

The 23k Ram Disk adds only two commands to Basic:

```
& = save
\ = load
```

There is no scratch command since each time the save is used it writes over whatever was previously saved.

When used in program mode it works in the same way as loading from within a program using disk or tape; the second program should be smaller than the first, and once loaded, the second program will start to run automatically and all variables and strings from the first program will remain. In direct mode it is useful for program development, allowing fast storage and retrieval of programs or source code, making it easy to check a routine from another program, look at a directory or to temporarily save the current program while another is RUN. It works great as an UNDO feature when working on a Basic program, although it can't UNDO a crash.

How it Works

The RAM DISK diverts the IGONE vector at $308/$309 and uses the RAM underneath the KERNAL, INPUT/OUTPUT and BASIC ROMS from $A000 to $FD00 for storage. The program saves whatever is in Basic memory, between the start of Basic (44/45) and the start of Variables (45/46). First the size of the program is calculated by simply subtracting the start of Basic from the start of variables – if the program is too large, the border colour will change, as a warning, and the program will not be saved. The length of the program is then subtracted from the top of the RAM DISK storage area ($fd00) and the result is the starting address at which the program is saved; the end of the saved program is always at $fd00, regardless of length. Programs are always loaded back into the address pointed to by the start of BASIC (44/45) and the start of variables pointer will be set to point to its end.

The RAM DISK program resides at the top of Basic Ram from $9EE8 (40680) to $9FFF (40959) so will not be compatible with any programs that use this area or the Ram under the Kernal. The programs storage capacity can be easily changed, by changing one number in the loader programs' DATA statements or by POKE–ing the number of blocks into 40723, to any size up to a maximum of 93 blocks. If there is a machine language program between $C000 and $CFFF, the DOS wedge for example, the maximum blocks should be 44 to ensure it's not overwriten by the RAM DISK. With a 44 block maximum, the memory from $D000 to $FD00 is used for RAM DISK storage and the memory from $A000 to $CFFF is free for other programs. Another example might be a revised version of Basic in the RAM between $A000 and $BFFF, in which case a 60 block maximum would be safe to use. The longer the program is, the closer to $A000 it will come. During the loading and saving process, interrupts are stopped and all ROM memory is switched for RAM. In order to try to conserve memory, the program uses a Basic loader that pokes the ML into memory, lowers the top of Basic and runs the program.

Compatibility

Zero page memory from $FB to $FE and $01 is used during loading and saving but these locations are not changed because they are stored temporarily in the RAM above $FD52 and replaced after the Load or Save. The reason for this is to enable the RAM DISK to reside with other programs that use these popular locations. Another effort at being compatible is the use of the vector $0308/$0309, which is copied and moved to the top of the program so that if another program is in memory and using this vector when the RAM DISK is started, there won't be a conflict or a crash. It's a good idea, therefore, to LOAD and RUN the RAM DISK after you have other programs installed. The program is compatible with the DOS wedge and most BASIC extensions that reside at 49152 or the start of basic, including the FAST ASSEMBLER (compute!'s gazette) on which it was written and the popular PAL assembler.

Word of Caution

The load (\) command should never be used when the pro-

gram is first started because it will cause the computer to crash as it tries to load a non–existent program. Therefore its a good idea to save the RAM as soon as the program is started. If you change the 93 in line 1050 of the loader program, the checksum will add up to the wrong amount and stop the program with an error message. Either remove the checksum (line 130), change its value or POKE 40723 with the number of blocks after the program is loaded. A program to be saved must be at least one byte less than the maximum allowable blocks.

Anyone that read the May '86 Bits and Pieces, Corrupting RAMTAS update, will have guessed why the RAM DISK only uses only 23.25k and leaves the top of memory (above $FD00) alone. The RAMTAS routine writes to the RAM at $FD30 to $FD4F each time the RESTORE key is hit and will destroy any program that was stored there.

Conclusion

As you can see, the RAM DISK works much like a disk or tape using program files. Its only drawback is that it can only hold one program at a time, but on the positive side it is simple, short and efficient and tries to be as compatible as possible with other software.

C64 RAM Disk: BASIC Loader

```
EB 100 rem* data loader for "ramdisk64" *
KK 110 for i=40680 to 40959:read a:poke i,a
DM 120 cs=cs+a:next i
HE 130 if cs<>41825 then print"!data error!":end
IH 140 poke 55,231: poke 56,158: clr
EG 150 sys 40680
CC 160 print"ramdisk–64 activated!"
PA 170 print" & to save"
NP 180 print" \ to load"
OL 190 end
MD 200 :
KE 1000 data 173,  8,  3, 141, 254, 159, 173,  9
FP 1010 data  3, 141, 255, 159, 169, 255, 141,  8
PG 1020 data  3, 169, 158, 141,  9,  3,  96, 160
CA 1030 data  1, 177, 122, 201,  38, 240,  3,  76
KC 1040 data 119, 159,  32, 115,  0,  56, 165,  46
OP 1050 data 229,  44, 201,  93, 144,  6, 206,  32
KP 1060 data 208,  76, 194, 159,  32, 206, 159,  56
GL 1070 data 165,  45, 141,  90, 253, 229,  43, 141
FP 1080 data  87, 253, 165,  46, 141,  91, 253, 229
FF 1090 data  44, 141,  88, 253,  56, 169,  0, 237
JM 1100 data  87, 253, 133, 253, 141, 252, 159, 169
AK 1110 data 253, 237,  88, 253, 133, 254, 141, 253
IL 1120 data 159, 165,  43, 133, 251, 165,  44, 133
PF 1130 data 252, 160,  0, 177, 251, 145, 253, 230
KE 1140 data 251, 208,  2, 230, 252, 230, 253, 208
GL 1150 data  2, 230, 254, 165, 251, 205,  90, 253
KJ 1160 data 208, 233, 165, 252, 205,  91, 253, 208
JN 1170 data 226,  32, 231, 159,  76, 194, 159, 201
ML 1180 data  92, 208,  71,  32, 115,  0,  32, 206
OP 1190 data 159, 173, 252, 159, 133, 251, 173, 253
ON 1200 data 159, 133, 252, 165,  43, 133, 253, 165
AF 1210 data  44, 133, 254, 160,  0, 177, 251, 145
KK 1220 data 253, 230, 251, 208,  2, 230, 252, 230
FK 1230 data 253, 208,  2, 230, 254, 165, 251, 201
OK 1240 data  0, 208, 234, 165, 252, 201, 253, 208
EE 1250 data 228, 165, 123, 201,  2, 208,  14, 165
EL 1260 data 253, 133,  45, 165, 254, 133,  46,  32
PA 1270 data 231, 159, 108, 254, 159,  32, 231, 159
LD 1280 data  32, 161, 225,  76, 174, 167, 165,  1
CI 1290 data 141,  89, 253, 169, 127, 141,  13, 220
DB 1300 data 169,  0, 133,  1, 162,  4, 181, 251
MJ 1310 data 157,  82, 253, 202, 208, 248,  96, 162
GJ 1320 data  4, 189,  82, 253, 149, 251, 202, 208
MA 1330 data 248, 173,  89, 253, 133,  1, 169, 129
DE 1340 data 141,  13, 220,  96,  0,  0,  0,  0
```

C64 RAM Disk: PAL Source Code

```
FD 100 sys700
GN 110 .opt oo
OO 120 ;
CP 130 *=$9ee8
CA 140 ;
EK 150 chrget    =   $0073
JA 160 ciaint    =   $dc0d
PM 170 bankin    =   $01
FP 180 tempzero  =   $fd52
LN 190 tempbloc  =   $fd57
FN 200 tempreg   =   $fd59
AF 210 sov       =   $fd5a
CG 220 basicvec  =   $0308
DB 230 border    =   $d020
GG 240 ;
OD 250 ;*** change basic vector
HB 260          lda  basicvec
AL 270          sta  tempvec      ;move to end of program
HD 280          lda  basicvec+1
PE 290          sta  tempvec+1
EG 300          lda  #<ram
HI 310          sta  basicvec
EH 320          lda  #>ram
HK 330          sta  basicvec+1
AE 340          rts
EN 350 ;
OJ 360 ;*** main routine starts here
PI 370 ram      ldy  #$01
GK 380          lda  ($7a),y      ;look ahead of basic
PG 390          cmp  #"&"         ;check for character
NN 400          beq  r2
PB 410          jmp  lode
KC 420 r2       jsr  chrget       ;start of save
PC 430          sec
AM 440          lda  46
GN 450          sbc  44
```

```
KL  460          cmp  #93          ;check size
HO  470          bcc  r3
BN  480          dec  border       ;warn too large
FM  490          jmp  exit
NC  500 r3       jsr  zersa
PH  510          sec
EM  520          lda  $2d
LK  530          sta  sov
CO  540          sbc  $2b
ND  550          sta  tempbloc     ;get number blks low
PO  560          lda  $2e
MK  570          sta  sov + 1
NA  580          sbc  $2c
CJ  590          sta  tempbloc + 1;number blks high
JN  600          sec
FI  610          lda  #$00
GK  620          sbc  tempbloc     ;subtract blks
IJ  630          sta  $fd
CO  640          sta  strdisk      ;start address in ram disk
PD  650          lda  #$fd
AP  660          sbc  tempbloc + 1
DM  670          sta  $fe
IP  680          sta  strdisk + 1
IG  690          lda  $2b
IN  700          sta  $fb
PH  710          lda  $2c
PO  720          sta  $fc
NF  730          ldy  #$00
CN  740 start    lda  ($fb),y      ;source
IN  750          sta  ($fd),y      ;destination
GP  760          inc  $fb
LD  770          bne  s2
NA  780          inc  $fc
CB  790 s2       inc  $fd
LF  800          bne  s3
BD  810          inc  $fe
PC  820 s3       lda  $fb
ON  830          cmp  sov
EC  840          bne  start
DD  850          lda  $fc
PB  860          cmp  sov + 1      ;keep going to the sov
CE  870          bne  start
OJ  880          jsr  zerlo
FF  890          jmp  exit
JO  900 lode     cmp  #"\"         ;check for character
AD  910          bne  exit
NC  920          jsr  chrget       ;start of load
ON  930          jsr  zersa
AN  940          lda  strdisk      ;start address in ram disk
DF  950          sta  $fb          ;store in source
CN  960          lda  strdisk + 1
JO  970          sta  $fc
IE  980          lda  $2b          ;start of basic (low)
GI  990          sta  $fd          ;destination  (low)
PE  1000         lda  $2c          ;basic (high)
IG  1010         sta  $fe          ;destination (high)
PH  1020         ldy  #$00
LE  1030 stld    lda  ($fb),y      ;source
AN  1040         sta  ($fd),y      ;target
IB  1050         inc  $fb
GF  1060         bne  l2
PC  1070         inc  $fc
GC  1080 l2      inc  $fd
GH  1090         bne  l3
DF  1100         inc  $fe
DE  1110 l3      lda  $fb
IM  1120         cmp  #$00
LN  1130         bne  stld
FF  1140         lda  $fc
IH  1150         cmp  #$fd
JP  1160         bne  stld
DB  1170         lda  $7b          ;check for running prgrm
MA  1180         cmp  #$02
FC  1190         bne  prgm
FP  1200         lda  $fd          ;direct mode then set sov
EL  1210         sta  $2d
KI  1220         lda  $fe          ;sov
EB  1230         sta  46
GA  1240         jsr  zerlo
KH  1250 exit    jmp  (tempvec)
JB  1260 prgm    jsr  zerlo
OG  1270         jsr  $e1a1        ;basic load routine
NE  1280         jmp  $a7ae        ;back to interpreter
NL  1290 zersa   lda  bankin       ;switch out roms
AH  1300         sta  tempreg
OK  1310         lda  #$7f
AC  1320         sta  ciaint       ;kill interupts
FF  1330         lda  #$00
BK  1340         sta  bankin
FN  1350         ldx  #$04
OB  1360 zs1     lda  $fb,x        ;move to a safe spot
HP  1370         sta  tempzero,x
LB  1380         dex
DI  1390         bne  zs1
EG  1400         rts
JC  1410 zerlo   ldx  #$04
AJ  1420 zl1     lda  tempzero,x   ;put back
KL  1430         sta  $fb,x
HF  1440         dex
BL  1450         bne  zl1
FJ  1460         lda  tempreg      ;switch roms on
DC  1470         sta  bankin
EK  1480         lda  #$81         ;start interupts
EO  1490         sta  ciaint
IM  1500         rts
DE  1510 ;************ program variables
CL  1520 strdisk    .byte 0,0      ;start address in disk memory
JL  1530 tempvec    .byte 0,0      ;points to interpreter loop
KH  1540 ;
```

Amiga Dispatches

by Tim Grantham, Toronto, Ontario

Report from the Shadowlands

With the demise of TPUG Magazine and one of the best jobs I've ever had, I found myself forced to accept work for one of the major computer retailers, configuring and installing PC compatibles for Fortune 500 clients. It's a dreary business. You need a plug–in card for everything, and they're all expensive. A Sysdyne colour/monochrome/graphics adaptor, for example, costs over $300 and takes up a 3" by 4" printed circuit board. (Exactly the same functionality is provided by a $30 chip in the C–128.) By the time you've added enough cards for a PC XT to actually *do* anything, you've spent three grand, and you still don't have a monitor, much less any software to run on the damn thing. But you'll never see a C–128 in a Bay Street office - after all, how could any self–respecting computer sell for less than $1000? Must be a game machine. I have developed a grudging respect for such machines as the Toshiba laptops - they're fast, they're compact, and they don't look like they were designed by the Pentagon. And I have respect for such powerful MS–DOS software as Word Perfect and AutoCAD. We need the richness these programs provide to challenge the Amiga's capabilities (they can keep the prices).

But then, I have just as much respect for good plumbing. My daily exposure to the PC world has brought about a new appreciation of the Amiga. It's in a class of its own: not just another glorified cash register but a synergizing lens designed by artists, for artists.

The Amiga will find a place in business. It's encouraging to see the Mac gaining acceptance in the corporate world: now that the business software is in place, the executives are realizing that the graphic interface provides a built–in solution to what is an added (and worse, hidden) cost on the PC compatibles - operator training. And the Mac's success can only help the Amiga.

Meanwhile, us wild–eyed creative types are having a wonderful time with the best unknown computer in the world.

Software News

The most important news is, of course, the release (at last!) of 1.2 Kickstart and Workbench, and of the Sidecar. I'll get to the Sidecar later on in the hardware section.

At press time, 1.2 is to be released for about $15 (US) and includes substantial documentation of the new features. That's a bargain, considering the enormous improvements made - MS–DOS upgrades typically cost $85 (US). Charlie Heath, author of TXed, has as an excellent comparison review of Lattice C and Aztec C in the November issue of Byte. Now that 1.2 has appeared, Lattice has released a major upgrade of its compiler, and it addresses (no pun intended) every concern raised by Cheath, and then some. According to a Lattice announcement, the goodies include:

- A two–disk package that includes a bootable system disk to simplify installation.
- A greatly enhanced library with over 255 functions (over 100 functions more than the standard Amiga C):
- Faster pointer and integer math
- Faster IEEE floating point routines (at least 5x)
- Direct support of the Amiga's FFP format floating point library
- A full macro assembler
- And completely new, expanded documentation.

"While remaining compatible with previous software, we have extended the object file format to include base–relative addressing, and our linker will support base–relative addressing modes and pc–relative branches to target locations that might otherwise be out of range. Addressing modes may be freely mixed in a program." A Professional Developer's Package is also available and will include several development utilities, and the highly rated Metadigm Metascope windowing debugger. The prices are as follows:

	List	Upgrade
Lattice AmigaDOS C Compiler V3.10	$225	$ 75
Professional AmigaDOS C Compiler	$375	$225

And if you're still not happy with Lattice's new–improved version of Alink, John Toebes VIII, of Hack fame, and the folks at the Software Distillery have produced a compatible, fast linker called Blink. It's available on the Fish disks, and public networks.

Aztec, meanwhile, is beta–testing an upgrade to their C compiler that has a complete debugger and patches to run the current version under 1.2

In other news on the wires: True Basic has a run–time module available, to create stand–alone Basic programs. . . PTE (Professional Text Engine) from Vladimir Schneider of Montreal provides fully programmable keyboard, menu, and mouse control. . . Grabbit! can dump HAM pictures, and any other screen, to any printer in Preferences. Won't print sprites, though. . . If you run Online! from the CLI and add a * to the command line ("online! *"), the program will come up in interlace mode. I tried this trick with Scribble!, a product put out by the same company, and that works too. . . After many announcements, Flight Simulator for the Amiga has been released, with two modes (Cessna, Lear Jet), and the ability to open up windows with different views. Jet will offer the intriguing possibility of hookup via modem –– pilot your Amiga in a dogfight with a friend across the world. . . Aegis has bought v1.1 of Musicraft and is selling it under the name Sonix. . . Softeam is still struggling to produce their PC–ET IBM PC emulation program: it works, apparently, and multitasks under AmigaDOS, but is not fast enough. Many of the routines are currently being re–written in assembler to improve speed. . . Leader Board from Access has great golf graphics but Activision's Mean–18 is the greater challenge. . .

Aegis Draw runs faster if you have expansion ram. Aegis has also released Draw Plus. . . The Spenser Organization, (201–666–6011), is coming out with an APL interpreter for $300 (US). . . Brown–Wagh, the distributors of Scribble! and Analyze!, is distributing Publisher! for the Amiga, written by Northeastern Software. For $200 (US) you get multi–column layout, text justification with kerning, wordwrap, mixing of IFF files within text (including resizing and cropping). . . There have been persistent rumours that Ashton–Tate ported dBase III to the Amiga some time ago. The story goes that they wanted a large sum of money up front from CBM to bring it to market, and CBM refused. Now it seems that they are waiting for enough Amigas to be sold to justify bringing it out. Meanwhile, the Word Perfect people are reportedly working on a full port of their renowned PC program. They had completed a stripped–down version, but decided Amigans would want a sophisticated word processor. Availability will apparently be the first quarter of '87. . . Alfred Aburto reports that the 68881 chip is supported under 1.2, but wasn't under 1.1. However, True Basic still crashes with the 68020 CSA board installed. . . To use the vt100 emulation in Online! under WB1.2, you need to load and run "setmap usa0" first.

It is possible to use outside fonts in DeluxePrint, according to a Plinker who goes by the handle AHN769. Boot with DP, open a CLI, and assign fonts: to whatever disk contains the desired fonts. Then, load DPrint, and when the drive lights go out, put in the fonts disk. The fonts will load and are available even though the menus in Dprint will indicate the old ones.

Chessmaster 2000, also from Electronic Arts is getting good response from purchasers. One can view the board from any angle (in 2–D or 3–D), play through a library of championship games, and have the program analyze one of your own games after play. It is apparently possible to multitask with Chessmaster if you invoke it from a CLI (without using "run"), and then use PopCLI (see below) to get another CLI.

There have been vociferous complaints about the first version of Datamat, a highly touted relational database. Apparently, the AmigaDOS version is riddled with bugs, is difficult to use, and has dreadful documentation. Mark Callaghan of Transtime Technologies says, however, that free upgrades will be out by the time you read this that will fix bugs and provide Intuition support. A new manual is supposed to be out within the next six months. You can reach Mark at 716–874–2010.

I was never able to get the Transformer to run PCTalk, a fine shareware terminal program. Others have had difficulty with similar programs. But word comes that Qmodem v1.08 (and only v1.08) will work fully under the Transformer. It's still very slow, but you can use another MS–DOS program called SPEEDY.COM to speed up the screen output.

Superbase for the Amiga has arrived! It's on sale in England and should be available from Progressive Peripherals here soon. It is relational, makes full use of Intuition, and can use IFF data. The 'personal' version will sell for about $150 (US). The 'development' version will arrive later, have a command language and cost about $250. It is not copy–protected but uses a dongle(!). Progressive Peripherals is also selling Logistix, a programmable spreadsheet that can use the entire 8 megabytes of RAM available to the Amiga.

Lastly, the richness of public domain software for the Amiga continues to surprise and please me –– the Fred Fish disks, the Amicus disks, not to mention the wealth of freely distributable software on the boards and nets. Graphics, sound, programming languages, games, often of high quality, are pouring forth. A few notables: Blink and PopCLI, (the latter lets you get a new CLI at any time, by just pressing an ESC–key combination), from the Software Distillery; DJJames's Comm, a fine terminal program; Fixhunk, a program you can use on programs like the first version of Scribble! to deal with the problem of gadgets mistakenly loaded into expansion memory (see last issue's column); hi–res HAM graphics created with ray–tracing programs on mainframes and transferred; and psound, a shareware sampled–sound editor/player that rivals the commercial versions from Futuresound and Mimetics.

And Alonzo Gariepy's impressive KickBench utility - this lets you create a combined Kickstart/Workbench disk that's ideal for such applications as bulletin board systems that must re–boot after a power failure. Lonnie has created a patch to Kickstart that causes it to convert the Kickstart disk (during bootup) to a standard AmigaDOS disk. If you have copied

Workbench to your Kickstart disk (there's room only for a stripped–down version), you can place another of Lonnie's programs called "kick" in the startup–sequence. Kick will rewrite the boot sectors on the disk to change it back to a Kickstart disk so that, should the power go off, the machine will reboot and the whole sequence start over again. It's ingenious hacks like this that make life worth living.

Hardware News

Despite my disdain for things IBMish, the Sidecar is an impressive piece of hardware/software. It's a 256K PCompatible, upgradable to 512K and with a built–in 5 1/4" drive, it can run almost anything that can run on an XT, and display it in a window on the Amiga. There are indeed three slots (I'm glad the single slot idea was killed) where you can put a hard–card that can be partitioned and used by both machines; a port for another drive (5 1/4" or 3 1/2", usable by either the Sidecar or the Amiga, but not both at the same time); and two port extenders on the front for the mouse and joystick. Amiga expansion memory can apparently be placed between the two computers, partitioned, and used by both. Price in Canada will be under a $1000, and will probably include MS–DOS. (No keyboard port, more's the pity. Stick a display card in there, and voila!, Siamese computers!)

Genlock should also be out shortly, after much delay. The Brits couldn't wait and designed there own version: unlike the North American one, it connects to the parallel port as well as the RGBA port, can be software controlled and, according to Jez San (author of Starglider for the Atari ST), can cause the video signal to replace any colour on the Amiga's display. Only in England, eh? Pity.

Jay Miner, designer of the custom Amiga chips, has announced that Commodore is working on the next generation of the Denise and Agnes chips. They will be able to address the full 2 Meg of chip memory; will have a much higher, non–interlaced resolution; and the blitter will be able to move 4K blocks of pixels rather than the current 1K. These chips will be incompatible with current Amiga 1000 hardware, but AmigaDOS should only require a minor upgrade. It might be possible though, to attach them to the expansion bus on a plug–in card.

Which brings to one of the more intriguing stories floating around the nets. Commodore is eager to sell the Amiga custom chips to third–party hardware developers. A gentleman called Chris Barr, it appears, is engaged in serious negotiations with C–A, to manufacture an Amiga card for a PC - that is, an entire Amiga, complete with AmigaDOS in ROM and 4K of RAM, that would plug into a regular PC slot! The Sidecar in reverse - the Rumble Seat, maybe? An external drive would probably be required to load Amiga programs, but it may be possible to produce the card for well under $1000 (US).

Timbits

"Inside the Amiga", by John Thomas Berry, is essentially a programming–in–C–on–the–Amiga book, something I, for one, have been looking for to pull together the enormously diverse information about the Amiga. This does the trick admirably for $34.95 (Can.). . . The Amigaforum in Compuserve is as busy as ever, but my favourite is now the Amiga Zone on PeopleLink. The quality of the information there is almost as good as the much more expensive Compuserve, and there is just as much PD software, if not more. The sysops there, CBM*HARV, DJJAMES, CBM*STEVE, and AMICUS (John Foust) are friendly and unpretentious, and the Sunday night CO (conference) is always well attended. Furthermore, GSARFF and others are working on ACO (Amiga Conference). This is like VMCO on the Mac - during CO's, the faces of the people online are on the screen in caricature form and change expression as the conversation flows. . . Auto–config is in Kickstart so that Kickstart detects the extra ram and places the system tables there, thus freeing up more of the chip memory (this according to DJJAMES). There will also be a way provided to patch Kickstart: vector tables to routines can be adjusted to point to new routines in RAM.

Finally, Atari Corp. is going public and has issued a prospectus. The interesting thing here is that they state that 150,000 ST's have been sold, far less than the million or so machines previously claimed by Compute!, and essentially the same as the Amiga. However, the ST was released earlier, and the figure includes European sales that have only just begun to happen for the Amiga. The truth gets out eventually.

I would appreciate any comments or questions you may have about the topics discussed. I can be reached c/o The Transactor, or on Compuserve (71426,1646), or on PeopleLink (AMTAG).

News BRK

Submitting NEWS BRK Press Releases

If you have a press release which you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance.

Transactor News

Subscription Intersection Set

TPUG is now shipping The Transactor to its members. If you are a TPUG member, you probably recall getting two Transactor magazines last issue. This "overlap" situation has since been dealt with. Using a formula based on the total number of pages, your Transactor and TPUG subscriptions have been combined. The following chart shows how many Transactors with TPUG inserts you'll receive based on the remainder of both combined.

Remaining:	TPUGs									
#T.'s	1	2	3	4	5	6	7	8	9	10
1	2	2	3	3	4	4	5	5	6	6
2	2	3	4	4	5	5	6	6	7	7
3	3	4	4	5	6	6	7	7	8	8
4	4	5	5	6	6	7	8	8	9	9
5	5	6	6	7	7	8	8	9	10	10
6	6	7	7	8	8	9	9	10	10	11

If for some reason we managed to miss an overlap and you again receive two magazines, please let us know. There were less than 400 matches in the two mail lists, but slight differences in names or addresses would result in undetected matches. These are nearly impossible to find without a thorough eyeball search. So if you are still getting two, please call us or TPUG and the adjustment will be made.

Although we have a list of the combined subscriptions, you won't notice any change in the expiry date on your mailing label. Since we still need to keep track of the expiry of each subscription, we will probably need to make changes to our databases. Hopefully by next issue we'll have the labels adjusted too.

No More GLINKS

That's right, no more. We've just shipped the last of the G-Link C64 to IEEE Interface, and no more will be made.

Schedule IRQ

Most of you probably thought this would be the "Simulations and Modelling" issue. We interrupted our regularly scheduled programme to bring you this "Languages" issue due to the abundant supply of Languages type material we've received. We've also received very little material suitable for a Simulations and Modelling theme, so if you have something we might be interested interested in, send it in soon. We now return you to our regular programming, already in progress.

Late Note

The article "Blazin' Forth" in this issue is based on a Forth compiler actually called "Blazin' Forth" written by Scott Balantyne. The compiler would take eons to enter by hand so it isn't printed in the mag. It is, however, included on The Transactor Disk for this issue.

Superpaks from Digital Solutions

Version 2.0 of the software trio from Digital Solutions is now in production. The new packages include both the 64 and 128 versions on the same disk. Each 2.0 Pocket package will sell for $59.95 U.S. or $84.95 Cdn. A Superpak will include all three for $99.95 U.S. or $139.95 Cdn. The Pocket Dictionary is still $14.95 U.S., $19.95 Cdn. However, they won't be available from us until next issue.

Version 1.0 is still available, and at terrific prices! The 64 and 128 versions still come in seperate packages, and their prices can be found later in News BRK, or on the Mail Order card. But the real deal is the special price for all three. The C64 Superpak is $49.95 U.S. or $59.95 Cdn. C128 Superpaks are $59.95 U.S. or $69.95 Cdn. To top it off, we'll throw in the Pocket Dictionary program for free!

Free Transactor T's with Mag+Disk Subscription

For a limited time only, subscribe or renew to a combination magazine and disk subscription, and we'll send you a free Transactor T-Shirt! You save 29% off the magazines, 16% off the disks, and get a Transactor T worth $13.95 ($17.95 if you order the jumbo size!) The T-Shirts come in 5 sizes (red only), with a 3-color screen featuring Duke, our mascot, dressed in a snappy white tux, standing behind the Transactor logo done in yellow with black "3-D" borders. The screen was done using a special "super-opaquing" process that cost us quite a bit more than those decals that crack and fade. Mine has been through the wash at least 30 times now, and it still shows virtually no sign of wear due to "washing machine punishment".

Transactor Disk Price Increase

A subscription to 6 Transactor Disks remains at $45.00. However, the price of single order Transactor Disks has been increased from $7.95 to $8.95 each - another good reason to take advantage of the above offer!

Refund Policy

Should any product you order be defective on receipt, return it and we'll send you another for no additional charge. Recently we've had a few items returned because "it's not quite what I wanted". We will credit your account (less shipping and handling) for purchases of other Transactor products, but we ask that you please be sure you need things like Micro Sleuths or RAM boards since we can't refund your money. While we're on the subject, although we've never had a subscriber ask for one, there are no refunds on subscriptions.

Transactor Mail Order News

New Subscription/Mail Order Card

We should have thought of this one sooner. The old mail order card brought two main complaints from many of our readers: sending a cheque meant pasting the card to an envelope, and the open face design left things like credit card numbers exposed. Thanks to Fred Cusick of Newmarket Ontario, our new card eliminates these problems. It also means we can increase the type size a

little, add more products, and still have room for some short descriptions. However, if you're using the card to order, we still suggest you pull it out and cross–reference with the list below for more details.

When folding up the card, please be sure the proper address shows on the outside. If our Buffalo New York address is showing and you drop it in a Canadian mail box (or vice versa), we're not sure what will happen to it! We *are* sure it will slow things up for you. Also, please use tape to hold it together – staples may upset the delicate balance of nature at the post office and your card may become extinct.

■ Moving Pictures - the C-64 Animation System, $29.95
This package is a fast, smooth, full-screen animator for the Commodore 64, written by AHA! (Acme Heuristic Applications!). With Moving Pictures you use your favourite graphics tool to draw the frames of your movie, then show it at full animation speed with a single command. Movie 'scripts' written in BASIC can use the Moving Pictures command set to provide complete control of animated creations. BASIC is still available for editing scripts or executing programs even while a movie is being displayed. Animation sequences can easily be added to BASIC programs. Moving Pictures features include: split screen operation – part graphics, part text – even while a movie is running; repeat, stop at any frame, change position and colours, vary display speed, etc; hold several movies in memory and switch instantly from one movie to another; instant, on-line help available at the touch of a key; no copy protection used on disk.

■ Volksmodem 12, w/cable, and CIS Intro-Pack, $329.00 Cdn., $199 U.S.
Not only do you get the Volksmodem 12 (DOC approved), but you get the cable at no extra charge (the C64 cable goes directly onto the User Port, and the RS232 cable is for any standard RS232 DB-25 female connector) Plus you'll receive a free CompuServe Intro-Pak which contains a User ID, a Password, and $15.00 of connect time! The Volksmodem 12 will work at 300 or 1200 baud, and is "Hayes compatible" so it will work with virtually any terminal software because the commands are controlled by you from the keyboard – just type "AT" (for ATtention) and follow with any of several easy-to-remember commands – no special POKing or elaborate dialing routines necessary! (I've been using a Hayes for almost 3 years, and my Volks for over a year - I love them both! - KJH) It comes with (get this) a 5 year manufacturer's warranty on parts and labour! The modem is shipped insured via UPS at no extra charge.

■ Intelligent I/O Interface Cards
■ BH100 I/O Interface Card w/documentation $129 U.S., $199 Cdn
■ BH100–AD8 8-Channel A to D Conversion Module $45 U.S., $69 Cdn
■ BH100 Beginners Course $159 U.S., $239 Cdn
■ BH100–S Security System $25 U.S., $39 Cdn
These products from Intelligent I/O will make great Christmas gifts! And if you've been wondering what to do with that VIC 20 that doesn't get much attention anymore, they're perfect! If you've ever wanted to start doing some real world interfacing, real easy, and inexpensively, then these items are ideal. The boards they sent us for evaluation are currently watching for floods in my basement. Too bad I didn't think of it before the flood – it only took about an hour using spare parts I had lying around – no resistors, no capacitors, just two strips of metal, a piece of styrofoam, a brick, and about 20 feet of wire that was also collecting dust. Once I get time, I intend to make it do some more surveillance since only one channel is currently in use. And the program to do it? A quick and messy 5 lines! Since the boards are memory mapped through the cartridge port, a PEEK is all you need! The 22 page manual is clear and concise. All products come with a 90 day manufacturer's warranty. Shipped insured via UPS at no extra charge.

■ Transactor T-Shirts, $13.95 and $17.95
As mentioned earlier, they come in Small, Medium, Large, Extra Large, and Jumbo. They're 13.95 each, $17.95 for the Jumbo. The Jumbo makes a good night-shirt/beach-top – it's BIG. I'm 6 foot tall, and weigh in at a slim 150 pounds – the Small fits me tight, but that's how I like them. If you don't, we suggest you order them 1 size over what you usally buy. The design is screened using a "super-opaquing" process so they wear much longer than your ordinary screens and iron-ons.

■ The Transactor Book of Bits and Pieces #1, $14.95
Not counting the Table of Contents, the Index, and title pages, it's 246 pages of Bits and Pieces from issues of The Transactor, Volumes 4 through 6. Even if you have all those issues, it makes a handy reference – no more flipping through magazines for that one bit that you just know is somewhere. . . Also, each item is forward/reverse referenced. Occassionally the items in the Bits column appeared as updates to previous bits. Bits that were similar in nature are also cross-referenced. And the index makes it even easier to find those quick facts that eliminate a lot of wheel re-inventing.

■ The Tr@ns@ctor 1541 ROM Upgrades, $59.95
You can burn your own using the ROM dump file on Transactor Disk #13, or you can get a set from us. There are 2 ROMs per set, and they fix not only the SAVE@ bug, but a number of other bugs too (as described in P.A. Slaymaker's article, Vol 7, Issue 02). Remember, if SAVE@ is about to fail on you, then Scratch and Save may just clobber you too. This hasn't been proven 100%, but these ROMs will eliminate any possibilities short of deliberately causing them (ie. allocating or opening direct access buffers before the Save).

■ The Micro Sleuth: C64/1541 Test Cartridge, $89.95 US., $129.95 Cdn.
This cartridge, designed by Brian Steele (a service technician for several schools in southern Ontario), will test the RAM of a C64 even if the machine is too sick to run a program! The cartridge takes complete control of the machine. It tests all RAM in one mode, all ROM in another mode, and puts up a menu with the following choices:

1) Check drive speed
2) Check drive alignment
3) 1541 Serial test
4) C64 serial test
5) Joystick port 1 test
6) Joystick port 2 test
7) Cassette port test
8) User port test

A second board, that plugs onto the User Port, contains 8 LEDs that lets you zero in on the faulty chip. Complete with manual.

■ Inner Space Anthology $14.95
This is our ever popular Complete Commodore Inner Space Anthology. Even after a year and a half, we still get inquiries about its contents. Briefly, The Anthology is a reference book – it has no "reading" material (ie. "paragraphs"). In 122 compact pages, there are memory maps for 5 CBM computers, 3 Disk Drives, and maps of COMAL; summaries of BASIC commands, Assembler and MLM commands, and Wordprocessor and Spreadsheet commands. Machine Language codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more. . . about 2.5 million characters total!

■ The Toolbox (PAL and POWER) $79.95
PAL and POWER from Pro-Line are two of the most popular programs for the Commodore 64. PAL is an easy-to-use assembler (most assembler listings in The Transactor are in PAL format), and POWER is a programmer's aid package that adds editing features and useful commands to the programming environment. They come with two nice manuals, and our price is $50 less than suggested retail!

■ AX1000 Amiga 1 MEG RAM Box $729.00 (+ $100 S&H) U.S.,
$1035.00 (+ $25 S&H) Cdn
■ AX2000 Amiga 2 MEG RAM Box $899.00 (+ $100 S&H) U.S.,
$1276.00 (+ $25 S&H) Cdn
The AX2000 adds 2 Megabytes of "fast" RAM to the Amiga, allowing more tasks to run in the system at once, or for use as a fast RAM-drive. The unit plugs into the expansion connector on the side of the Amiga and duplicates the connector for other devices to plug into. Up to two RAM boards may be plugged in together (limited by the Amiga'a power supply), adding 4 Megabytes. The box has "auto-config", so with Kickstart 1.2 the RAM will automatically be added to the system when it is booted. If you are using Kickstart 1.0 or 1.1 (no auto-

config), you can use the program included with the AX2000 to add the memory to the system, and change your startup-sequence to automatically add the memory on power-up. Standard expansion bus architecture was used in the design of the AX2000, ensuring compatability with all peripherals and operating system releases. The unobtrusive steel box is the same height and colour as the Amiga, and snugs up to the side without taking up much extra space. The unit is built tough and comes with a 1 year manufacturer warranty.

This seems to be the most highly-recommended Amiga RAM board, and the first one to actually be available, so we're selling it here at The Transactor. You can order the AX2000 or the 1-Meg AX1000 from the subscription form in this issue. Shipping and Handling to the U.S.A. is via courrier and includes all customs clearance, or you can opt to clear shipments yourself and have it shipped "collect".

■ Superpak 1.0	C64	$49.95 US, $59.95 Cdn
■ Pocket Writer	C64	$29.95 US, $39.95 Cdn
■ Pocket Planner	C64	$29.95 US, $39.95 Cdn
■ Pocket Filer	C64	$29.95 US, $39.95 Cdn
■ Superpak 1.0	C128	$59.95 US, $69.95 Cdn
■ Pocket Writer	C128	$39.95 US, $49.95 Cdn
■ Pocket Planner	C128	$39.95 US, $49.95 Cdn
■ Pocket Filer	C128	$39.95 US, $49.95 Cdn
■ Pocket Dictionary		$14.95 US, $19.95 Cdn

As mentioned earlier, you can now get all 4 programs from Digital Solutions for one low price! In fact, even if you average the price of all four, it comes to less than the price of two!

■ The TransBASIC Disk $9.95
This is the complete collection of every TransBASIC module ever published up to Volume 7, Issue 01. There are over 120 commands at your disposal. You pick the ones you want to use, and in any combination! It's so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

■ Super Kit 1541 $29.95 US, $39.95 Cdn
Super Kit is, quite simply, the best disk file utility there is. No more losing those valuable copy-protected originals (like what's happened to me twice too many times). So far we've shipped over 600 Super Kits and orders continue to pour in.

■ Gnome Speed Compiler $59.95 US, $69.95 Cdn
This compiler is for BASIC 7.0 on the Commodore 128.

■ Gnome Kit Utility $39.95 US, $49.95 Cdn
Gnome Kit is a Commodore 128 utility with enhancements for the BASIC editor (like Trace, Find, Renumber, Delete, Auto, etc.) as well as enhanced monitor commands, and floppy disk monitor functions.

Transactor Disks, Transactor Back Issues, and Microfiche

All issues of The Transactor from Volume 4 Issue 01 forward are now available on microfiche. According to Computrex, our fiche manufacturer, the strips are the "popular 98 page size", so they should be compatible with every fiche reader. Some issue are ONLY available on microfiche – these are marked "MF only". The other issues are available in both paper and fiche. Don't check both boxes for these unless you want both the paper version AND the microfiche slice for the same issue.

To keep things simple, the price of Transactor Microfiche is the same as magazines, with one exception. A single back issue will be $4.50 and subscriptions are $15.00. The exception? A complete set of 18 (Volumes 4, 5, and 6) will cost just $39.95!

This list also shows the "themes" of each issue. "Theme issues" didn't start until Volume 5, Issue 01. The Transactor Disk #1 contains all program from Volume 4, and Disk #2 contains all programs from Volume 5, Issues 1–3. Afterwards there is a separate disk for each issue. Disk 8 from The Languages Issue contains COMAL 0.14, a soft-loaded, slightly scaled down version of the COMAL 2.0 cartridge. And Volume 6, Issue 05 published the directories for Transactor Disks 1 to 9.

■ Vol. 4, Issue 01	(■ Disk 1)
■ Vol. 4, Issue 02	(■ Disk 1)
■ Vol. 4, Issue 03	(■ Disk 1)
■ Vol. 4, Issue 04 – MF only	(■ Disk 1)
■ Vol. 4, Issue 05 – MF only	(■ Disk 1)
■ Vol. 4, Issue 06 – MF only	(■ Disk 1)
■ Vol. 5, Issue 01 – Sound and Graphics	(■ Disk 2)
■ Vol. 5, Issue 02 – Transition to Machine Language	(■ Disk 2)
■ Vol. 5, Issue 03 – Piracy and Protection – MF only	(■ Disk 2)
■ Vol. 5, Issue 04 – Business & Education – MF only	(■ Disk 3)
■ Vol. 5, Issue 05 – Hardware & Peripherals	(■ Disk 4)
■ Vol. 5, Issue 06 – Aids & Utilities	(■ Disk 5)
■ Vol. 6, Issue 01 – More Aids & Utilities	(■ Disk 6)
■ Vol. 6, Issue 02 – Networking & Communications	(■ Disk 7)
■ Vol. 6, Issue 03 – The Languages	(■ Disk 8)
■ Vol. 6, Issue 04 – Implementing The Sciences	(■ Disk 9)
■ Vol. 6, Issue 05 – Hardware & Software Interfacing	(■ Disk 10)
■ Vol. 6, Issue 06 – Real Life Applications	(■ Disk 11)
■ Vol. 7, Issue 01 – ROM / Kernel Routines	(■ Disk 12)
■ Vol. 7, Issue 02 – Games From The Inside Out	(■ Disk 13)
■ Vol. 7, Issue 03 – Programming The Chips	(■ Disk 14)
■ Vol. 7, Issue 04 – Gizmos and Gadgets	(■ Disk 15)
■ Vol. 7, Issue 05 – Languages II	(■ Disk 16)

Industry News

The following items, compiled by Astrid Kumas, are based on press releases recently received from the manufacturers. Please note that product descriptions are not the result of evaluation by The Transactor.

Sorry, Wrong Number

Arghg! Two wrong numbers actually. The phone number of SoftTools, page 80 of the last issue, should be (514) 739-3046. The number for Innovative Software, also on page 80, should be (215) 372-5438. Our apologies for any inconvenience these errors may have caused.

The MSD DOS Reference Guide

David Martin plans to release his first book publication on December 1, 1986. The book entitled, "The MSD DOS Reference Guide", is for MSD single and dual drive owners who are interested in exploring their drive's Operating System. The guide provides a fully commented RAM map and source code ROM memory map. The ROM maps consist of source code so that a programmer can follow the inner workings of the DOS much more easily. The book also provides some nifty programs (utilities, etc.) that are ready to key in and enjoy. A separate disk is available for folks that are too busy to type in the book's programs. It will also include a collection of MSD public domain utilities as an added bonus.

USA $20.00 Book $6.00 Disk $3.00 shipping (add $5.00 for COD)
Canada $30.00 Book $8.00 Disk $7.00 shipping

Make cheques payable to David W. Martin
1417 South Heron Drive
Seabrook, Texas, USA, 77586

New Services on QuantumLink

QuantumLink, a telecommunication service for Commodore computer owners across United States and Canada, has added several new options to their information and entertainment sections.

QuantumLink has recently added to its service a forum for users of GEOS. GEOS, (Graphic Environment Operating System), is an icon– and menu–based software program that gives a Commodore computer the look and feel of a Macintosh. The new interest group provides support via Question and Answer

sessions with the developers of GEOS (Berkeley Softworks), conferences to discuss GEOS applications, and news on the latest GEOS developments and software.

In an effort to allow QuantumLink members to visualize their online acquaintances via digitized photos, an online Photo Gallery has also been introduced. Special photographic equipment converts subscriber photographs into computer programs. Each program is then placed online in the Photo Gallery for other users to download and display.

The Mall, QuantumLink's shopping section, has been expanded to include additional discounted products and a live auction.

RockLink, QuantumLink's most entertaining addition, offers the latest news and information on all aspects of the music industry. Special guests, including musicians, writers and producers from around the world, appear monthly for online discussions in the Auditorium. RockLink offers a rock library of historical hits, a music review board for users to post their own opinions, a daily music trivia question, backstage gossip, information on recent video releases, concert dates etc.

Users who would like to subscribe for the QuantumLink services can obtain more information by calling 800-392-8200. QuantumLink's basic service costs $9.95 US a month and can be accessed with a Commodore 64 or 128 computer.

PaperClip II for C-128

Batteries Included have re-designed their best-selling wordprocessor PaperClip for use on Commodore 128. The new version, called PaperClip II, is now available for $99.95 Cdn. or $79.95 US. PaperClip II's new features include:

- multiple columns, reverse video scroll, chaptering;
- fast, integrated 30,000-word spelling checker;
- maximum document size expanded to 999 lines;
- built-in telecommunications module to access on-line services;
- compatibility with C-64 PaperClip text files.

In preparation is PaperClip Elite - PaperClip's version for the Amiga. It will be released some time next year. For more information, contact:

Batteries Included
30 Mural Street
Richmond Hill, ON
L4B 1B5 (416)881-9941

New Products for C-128 from Abacus

Abacus Software has announced the release of three productivity packages for the Commodore 128 computer.

SpeedTerm 128 is a command driven terminal program that can be used with most modems for the C-128. In addition to the standard options, it also offers support for Xmodem and Punter file transfer protocols, VT52 and VT100 terminal emulation with cursor keys, a 45K capture buffer and user definable function keys.

TAS-128 is a technical analysis system for stock market charting. Using TAS-128, the investor can automatically download indicators from DJN/RS or Warner and then build a variety of charts on the split screen: 7 moving averages, 3 oscillators, 5 volume indicators, comparison charts, trading bands, least squares and others. The user can also take advantage of such features as automatic and unattended logon, fast-draw charts using up to four windows, and macro capabilities.

The last of the newly released products is PPM-128, an upgraded C-128 version of Personal Portfolio Manager for tracking the performance of stocks, bonds or options.

Suggested retail price for each of these products is $59.95 US. For more information, contact:

Abacus Software
2201 Kalamazoo S.E.
P.O. Box 7211
Grand Rapids MI
49510 (616)241-5510

Extend-A-Key

Extend-A-Key is a small device from Cox Enterprises. It should come handy to users who upgraded from C-64 to C-128 computer, and at the same time own a lot of C-64 software. Many old copyright protected programs operate only with an accompanying key made for the C-64. The same key most often does not fit into the C-128. Extend-A-Key allows for a simple retrofit of old keys and old software to the new C-128. The device plugs directly into the joystick port on the C-128.

The price for Extend-A-Key is $6.95 US including postage. The orders should be sent to:

Cox Enterprises
883 S.E. Bethel Place
Corvallis OR
97333

Aegis Art Pak, Volume 1

There is a new addition to Aegis' line of graphics software for the Commodore Amiga.

The Art Pak series is designed to provide the Amiga user with precreated art. It can be used with the Aegis Images professional paint program, Aegis Animator, and with Aegis Draw, the entry-level CAD program for the Amiga. Art Paks retail for $34.95 US each.

Volume 1 includes photograph quality artwork of buildings, which can serve as backdrops, or as pieces of cel animations for creating one's own moving animations. For additional information, contact:

Aegis Development, Inc.
2210 Wilshire Blvd. #277
Santa Monica CA
90403 (213)306-0735

File Archive Utility for the C-64

ARC, a new product from Ampere Metal, creates file archives using data compression techniques. It allows the user to combine any number of related files into a single archive file which is generally 20 to 60 per cent smaller than the combined lengths of the original files. Archives can later be dissolved to obtain exact duplicates of the contained files.

ARC is available as an extension to the BASIC interpreter and adds several direct mode commands including a simple text editor and an MS DOS like disk interface. ARC supports multiple drives. An 80-column version is available for the Batteries Included BI-80 adaptor.

ARC sells for $20.00 US, and can be ordered from:

Ampere Metal
80 Hale Road, Unit 4
Brampton, Ontario
Canada, L6W 3M1.

TPUG News

Volume 1, Number 2

President's Report

In the past year, there have been some significant changes to the operation of the club. Some of those changes that have taken place, or are in progress, include:

We have moved again. our new address is 5300 Yonge Street, which is right on the TTC route about 1 mile north of the 401. Our rent is now $1000 per month, down from $3000 per month at the Duncan Mill address. We also have a little more space. While we are still in the basement as before, our usable space is now 2,000 square feet instead of 1,500 square feet.

As of September of this year, we have discontinued producing our own publication, TPUG Magazine. Instead, we have contracted the Transactor to produce a magazine for us. This will consist of the normal Transactor magazine – about 80 pages of excellent material 100% advertising free – as well as an eight–page TPUG supplement produced just for TPUG. The reason for this is simple. We could no longer afford the luxury of our own glossy magazine at the present size of membership (8000+). We also have not been successful in getting adequate advertising revenue for TPUG Magazine to ease the burden. Moving to the Transactor will cut our costs in half, while at the same time provide our members with a quality magazine.

Since most of the financial work is being handled by our accountant, we have reduced staff in the TPUG office. All of the magazine staff and some of the office staff were laid off. We now have two people, one full–time, one part–time, in the office to handle such things as meetings, disk copying, answering mail, etc. Our operating costs will now be more in line with the present size of the club.

We have added meetings for some of the new machines produced by Commodore. The Amiga and C–128 meetings have been going on for about a year now and are in full swing with a large number of new disks added to our library.

We have also produced a new VIC 20 and C–64 catalogue. The C–64 catalogue appeared in the first edition of the TPUG/Transactor insert.

After two years of losing money, TPUG made a slight profit of $6503 last year. With the reduced cost of the magazine, office rent and salaries, we should also do quite well this year.

Last year we ran the Commodore SIG on CompuServe for several months. However, after our contract expired, we lost that contract to an ex–board member, Louise Redgers. We are still up and running on Delphi, which helps us serve our members all over the world.

I would like to thank the many volunteers who, over the years, have helped to make TPUG what it is today. Some of these people include:

Our meeting co–ordinators: Donald Dalley, Victor Gough, John Easton, Allan Farquharson, Gerry Gold, Avy Moise, Anne Gudz, Don Farrow, Jim Hamilton, Keith Falkner, Gord Campbell, William Barrett, George Skinner, Mike Donegan.

Our librarians: Mike Donegan, Keith Falkner, Gord Campbell, Adam Herst, Colin Justason, Bill Dutfield, George Davis, Derick Campbell, Victor Gough, Richard Best, Paul Atchison.

Our Board of Directors: Carl Epstein, Gerry Gold, John Easton, Anne Gudz, David Bradley, Richard Bradley, Keith Falkner, Mike Donegan, Gary Croft.

Our BBS operators: Sylvia Gallus, Steve Punter

I would like to close by thanking all of you for your support throughout the last year, and wish you luck and continued success in the years to come.

Chris Bennett, TPUG President 1985–1986

TPUG BBS

telephone number:

(416) 273–6300

Operating hours:
24 hours per day
7 days per week
The password is. . .

NOEL

TPUG MEETING SCHEDULE

Sept. 1986 to June 1987

All meetings begin at 7:30 pm sharp, unless otherwise specified. Capitalized dates indicate that the meeting does not fall on its normal day of the month.

VIC 20 Chapter: York Public Library, 1745 Eglinton Ave. W. (just east of Dufferin) in the Story Hour Room on the second Tuesday of the month unless otherwise specified.
1986: Dec 19
1987: Jan 13, Feb 10, Mar 10, Apr 14, May 12, June 9

Commodore 128 Chapter: York Public Library, 1745 Eglinton Ave. W. (just east of Dufferin) in the auditorium on the first Tuesday of the month unless otherwise specified.
1986: Dec 2
1987: Jan 6, Feb 3, Mar 3, Apr 7, May 5, June 2

COMAL Chapter: York Public Library, 1745 Eglinton Ave. W. (just east of Dufferin), in the Story Hour Room on the fourth Tuesday of the month unless otherwise specified.
1986: DEC 18
1987: Jan 22, Feb 24, Mar 24, Apr 28, May 25, June 23

Amiga Chapter: Clarkson Secondary School, Bromsgrove, just east of Winston Churchill Blvd., Mississauga; at 7 p.m. in the Little Theatre on the third Thursday of the month, unless otherwise specified.
1986: Dec 18
1987: Jan 22, FEB 13, MAR 13, Apr 16, May 22, June 19

Westside Chapter: Clarkson Secondary School, Bromsgrove, just east of Winston Churchill Blvd., Mississauga; in the Little Theatre on the third Wednesday of the month, unless otherwise specified.
1986: Dec 18
1987: Jan 22, FEB 13, MAR 13, Apr 16, May 22, June 19

Brampton Chapter: Brampton Public Library, Four Corners Branch, 65 Queen St., on the second Thursday of the month, unless otherwise specified.
1986: Dec 11
1987: Jan 8, Feb 12, Mar 12, Apr 9, May 14, June 11

68000 Chapter (formerly SuperPET): Curtis Lecture Hall C, York University Campus (on the north side of the ROSS Building), on the third Wednesday of the month unless otherwise specified.
1986: DEC 10
1987: Jan 21, FEB 11, Mar 18, APR 8

Commodore 64 Chapter: Auditorium, York Mills Collegiate Institute (on the north side of Wilson Ave. between Bayview and Leslie), on the last Monday of the month unless otherwise specified.
1986 - DEC 15
1987 - Jan 26, Feb 23, March 30, Apr 27, May 25, Jun 29

Eastside Chapter: Dunbarton High School (go north on Whites Rd. from the traffic lights at Highway 2 and Whites Rd. to next traffic lights; turn left to parking lots), on the second Monday of the month unless otherwise specified.
1986: Dec 8
1987: Jan 12, Feb 9, Mar 9, Apr 13, May 11, JUNE 1

Total Software Development System from Piratebusters

Assembler, editor and programming utilities for Commodore 64 and Commodore 128

Review by Chris Zamara

At first glance, TSDS appears to be just a collection of standard utilities: an editor enhancement utility a la BASIC aid, an assembler, a sprite editor, a character editor, and a sound effects generator -- the kind of thing to make a jaded programmer yawn. After using the package for a short while, however, it becomes obvious that the programs in TSDS are the product of a programmer who has created high quality tools for his own use, and honed those tools to a keen edge.

The TSDS package consists of a manual and a single disk. The manual is 27 pages long and coil-bound. The software comprises several programs, the main one being the editor/assembler system (sysedt and sysasm). Also on the disk are a sprite editor (sped), a sound effects editor (sed), a character/matte editor (macsed), an unassembler (unasm), and sample files for most of the programs, including assembler source code. There are also BASIC programs to convert source files from MAE or PAL assembler format into sysasm format.

The manual is well written, though fairly brief. For TSDS's intended audience of programmers, the brevity of the manual is a blessing, since it makes an excellent reference book. Generally, programmers don't want tutorials, they want lists and descriptions of functions. All functions are clearly laid out in the table of contents and explained in a concise paragraph. There are separate sections for each of the programs in the TSDS package, and bold type makes the important reference information stand out from the text.

Turning to the disk: aside from its actual content, an excellent feature right off the bat is that no copy protection is used. That means that you can copy the programs onto your standard utility disk, saving you the hassle of always breaking out the special TSDS disk. It also means that you'll always have a backup if something goes wrong. TSDS loads quickly (it uses a fast-load), won't bash your drive's head into a frenzy, and can be used on drives other than a 1541. It also won't wipe out your disk or print a message wishing you a horrible death if it thinks you're trying to make a copy of it. Any unprotected program is much more useful than a protected one, so Piratebuster's decision to go this route is commendable. And, as the name of the company suggests, they will thwart the software pirates by not giving them any protection to break. The only thing they ask in return is that you pay for your copy of the program. Sounds like a fair deal to me.

Except for the assembler, none of the other programs are integrated with the TSDS editor, and must be loaded and run separately. The core of the package is the editor/assembler; the rest of the disk is a collection of extra programs, seemingly thrown in as a bonus. Because of this isolation, each program will be treated separately below.

SYSEDT: The Editor

When you load TSDS (or boot the disk on the C-128), the sysedt program comes up, along with the built-in assembler. Sysedt makes the standard program-editing environment a better place to work in, especially if you are writing in assembler. There is a special editing mode for assembler source, and the edit command can be used to select source (S) or basic (B) editing. In the source mode (the default), program lines are not tokenized or modified in any way as they are entered, so you are not hampered by BASIC-dictated restrictions when editing assembler files. For example, you are free to indent lines with any number of spaces for readability, and you can use question marks freely without having them show up as print when listed. Using the put command, source programs are saved as standard CBM-ASCII sequential files with no line numbers, like Commodore's assembler format. When retrieving files with the get command, line numbers are automatically added so that the file can be edited. This standard method of storing files means that you can edit source code with your favorite text editor if you wish, and otherwise manipulate them as you would any CBM-ASCII text file.

Another prominent feature of sysedt is its built-in 1541 fast load. On all loads and gets, the fast-loader is invoked if a 1541 drive is being used. Files are retrieved over three times faster than with a regular load. The system is smart enough to only attempt fast loading if a 1541 drive is connected, but if your drive isn't up to the stresses of warp speed, you can turn off the fast load with the slow command. The fast-load capability is sure nice to have, but saves are still as slow as ever.

Sysedt redirects the reset and restore vectors, which can be a real time saver. If you have a reset button, using it to get out of a crash doesn't mean losing your current source file or rebooting TSDS. You come back up with TSDS re-initialized, and with your border and background colour settings intact! Typing old will bring back any source you may have been editing before you hit the panic button. If you were able to get out of your crash with a run-stop/restore, your colours won't be messed up either.

Excepting these features, sysedt is much like Power or BASIC aid in that it allows you to scroll forward and backward through your program using the cursor keys, adds some handy commands to BASIC, and allows you to redefine any key to print a character string of your choice.

The program scrolling feature is pretty slick, allowing you to position the cursor on any column during the scroll, and letting you select the scroll speed. The scroll is designed to work nicely with the special list command, which doesn't print break or ready after the listed lines. After the list, moving the cursor down will display further program lines with no breaks in the listing. The only awkward part of the program scrolling is that once you scroll to the end of the program, you can't continue to push the program up the screen by cursoring down; to make room for typing you have to switch to return or shift/return. In other programs, I would call that an annoying quirk, but after spending some time with TSDS, I can't help thinking that Kevin Pickell (the author) purposely made the scroll work in precisely the manner it does, for some reason I just don't appreciate yet. That is the extent to which this system has been thought out.

The commands that have been added to BASIC to aid in editing have been kept to a mercifully small number; a system that requires you to memorize a hundred commands certainly can't do much for productivity. There is one command in here that is sadly lacking in many BASIC enhancements, and it's my favourite: help. You can get a list of all commands at any time by just typing the magic word. Most of the other commands are equally intuitive. The familiar find, change, renumber, auto and delete commands are all here, but often with high-powered extra capabilities you don't find elsewhere. For example, find and change let you search for a string occurring at the beginning or end of a line, search for an exact match, search only outside of quotes, and so on. All commands that use line numbers as parameters, including list, allow labels to be used instead. You could list a subroutine, for example, with the command list blowup-. And here's a good idea -- any command preceded by an asterisk will direct its output to the current printer device (normally 4) instead of the screen.

Users of the DOS wedge utility will find that TSDS supports the single-character commands they are already familiar with: >, / and the rest. The exec command lets you execute sequential files as a series of commands, or turn SEQ files into BASIC programs. When TSDS comes up after it is loaded, it execs a file called ute. This file contains some handy key definitions and displays them. You can edit this file to your own specifications.

The key command is used to define a keystroke to print out any string of characters. Touching a single key can spit out a commonly-used command, filename or editing operation. You can define any key, but some keys seem to start repeating as soon as you hold them down, requiring a skilful quick tap on the key to use. A rather severe limitation is that you can only define up to 15 keys, with a total of 255 characters for all keys. Also, there is no way to program a key to execute a BASIC or machine code routine. This feature, which is found in Brad Templeton's Power utility, would really help for setting up editing functions. In short, the programmable key capabability of TSDS is a definite plus, but it could go further.

SYSASM: The Assembler

The TSDS Assembler is something between a convenient to use but limited assembler, such as PAL, and a less convenient but full-blown linking assembler like the Amiga's.

The assembler is invoked from sysedt with the assemble or asm command. You specify the source file and optional object file in the asm command, then sysasm takes over. It is a full-featured MOS-standard two-pass macro assembler, and is quite fast (I compiled a 24 block source

file in about 13 seconds using an 8050, and 40 seconds with a 1541). After compilation, you return to the sysedt environment, with a listing of assembly errors as the current source file in the editor. If you were working on something before the assembly that you hadn't saved, too bad. This file handling scheme is quite a shock for programmers raised on PAL, where a program is assembled directly from BASIC text space, the object goes to memory, and errors are printed as encountered. For those programmers, remember this when using sysasm: "Toto, I don't think we're in Kansas any more".

Actually, assembling from a disk file has its advantages. The .lib pseudo-op lets you include another source file in the assembly, so a good approach for a long program is to build it in modules, using a main module that just strings all of the other modules together with .libs. In that case, you want to always assemble the main module, which probably won't be what you have in memory anyway. Just make sure that what is in memory is saved before you assemble.

Sysasm can print the list of symbols from the program, sorted numerically or alphabetically, or unsorted. There are quite a few pseudo-ops, ranging from standard ones such as those used for conditional assembly, to more unusual but useful ones for generating checksums and scrambling the object code. An interesting form of .byt is .pok, which converts ASCII characters to their screencode equivalents. Bin is like byt, but allows you to use a disk file to supply the byte values -- great for including sprite definitions, music data and so on with object code. Additional pseudo-ops are included to make sysasm more compatible with PAL, though PAL's .bas, for example, is not supported.

There is no linker included, so sysasm falls short of being a full-fledged assembler. You can't make references to any labels that aren't included in your source file (with the .lib files added in). In this way, assembly is made simple, since linking is not required, but for large programs, an assembler with a linker might be better. However, sysasm is better than PAL for large programs, mostly because of the .lib command and macro capability. If you have PAL and prefer to use it for quick assemblies from memory, you can run it along with TSDS! That way you have the best of both worlds.

As with sysedt, the assembler is full of thoughful features. For example, attempting to abort an assembly by pressing the stop key brings up a prompt asking if you want to quit or continue. This allows an elegant exit, closing all files properly. A small point, perhaps, but it illustrates the care that seems to have gone into each part of the system.

The Separate Programs

Although separate, the sprite, sound and character editors can all be used from within the sysedt environment. You can activate them with the up-arrow (load and run), and you can exit from them with the stop key, coming back into the undisturbed sysedt environment. Not complete integration, but it's better than having to reset and reboot TSDS.

SPED: The Sprite Editor

The sprite editor is menu-driven and supports features beyond just the creation of sprite definitions. Up to 255 sprites can be defined, with the current sprite page displayed for editing, and the sprite itself displayed below. Any sprite page can be defined, and switching from one page to another is instantaneous. The menu has options to animate sprites, change colours, fill pages with rotated versions of a sprite, combine two sprites, load and save sprite definitions, manipulate a sprite by moving, flipping and rotating, transfer sprite pages, and view 64 sprites at a time. Multicolour sprites are also supported, and multicolour mode is automatically invoked when you switch to a multicolour sprite page. Up to six sprites can be combined in the sprite display using the overlay feature, for images consisting of a combination of several sprites.

This sprite editor can probably replace whatever you're using now, and be easier to use, faster and more powerful. It is definitely a worthwhile program to have in your collection. The only thing that detracts from its ease of use is the fact that the menu lists the single-key options, but has no descriptions of their function. Executing a function is often a hit-and-miss proposition. When you have hundreds of programs in your software library, and you use most of them only once every few months, you don't want to have to memorize anything for the next time. At least the control keys are listed on the menu -- some programs don't even go that far.

SED: The Sound Editor

Sed brings up a screen full of cursor-operated slide controls that let you vary the SID chip's major parameters: attack, decay, sustain, release, frequency, waveform and pulse width. An *envelope follower* -- another amplitude envelope -- can also be shaped via slide controls to create a wide variety of effects. An *intensity* control, which varies the speed of the envelope follower, is also provided, and the number of times to repeat the effect (up to 15) can be selected as well. Up to 255 such screens of effects can be defined, and they can be given names, which appear at the top of the screen. Effects can be loaded and saved as with the sprite editor. Loading is accomplished without having to type in a filename: all sed sound effects files on the disk are displayed, and the desired one is chosen with the cursor keys. (The sprite and character editors work in the same way.)

A file of sample sound effects is included on the disk to show some possibilities. The variety of effects possible with just a few controls is quite amazing. Once you've saved your sound creations, you can broadcast them from your own programs by using a source file included on the TSDS disk. By assembling the file using your sound data, you will create a machine language program that allows you to play any of your effects at will.

Using this package must be the simplest way to create sound effects, since you basically just twiddle knobs (actually, move sliders) until it sounds right, then make as many such effects as you desire. Not all features of the SID chip are utilized, though, and there is no polyphonic capability allowing you to combine sounds, so the more sophisticated SID-manipulations are out.

MACSED: The Character Editor

This is a unique program, but one for which there is a real need when designing character-based graphics screens. The program is actually a tight integration of a character editor and a pseudo-text editor that lets you put together characters in any pattern. Like the sound effects editor, this one is easy to use even without the manual, after looking at the screen for a while to figure out what everything does.

The 'matte editor' portion of the program is much like a word processor, allowing you to lay out your custom characters to create a graphics screen. A cross reference of which characters map to which keys is maintained at the top of the screen to help you choose your characters. The display will scroll to the right up to 255 columns, depending on how you set the 'window size'. Within the current window, you can create any image that can be made up from the existing characters. This character matte can then be saved.

From the matte editor you can select the character editor. The character editor shows a group of 64 characters, a character editing field, and the current character in the field displayed in all 16 colours. You can pick any character in the field and edit it, and copy or transfer ranges of characters. Multicolour character mode is also supported; as usual, Kevin Pickell didn't miss a trick. Character sets can be saved independently from mattes. It's easy to switch between editing characters and mattes so that you can build a new character whenever you need it in the matte.

While the need for this program is obviously quite limited, when you do want to create character-based graphics screens, it's invaluable. I had to write my own quick'n'dirty matte editor once, and the time I would have saved by using this one instead would pay for a TSDS package. (If you want an attractive title screen for your program or you want to write a game, consider using macsed to create a character matte instead of going to hi-res. Character screens load faster, use less memory, and let you move things around faster and easier. You can also smooth-scroll text screens vertically or horizontally.)

The TSDS System

Taken as a whole, the TSDS package is a great bargain. Even if you just need a program development tool and have no use for the extra programs, the editor/assembler alone is worth the price of the TSDS package. If you develop programs mostly in assembler, it may be worth considering switching over to TSDS from whatever you're using now. BASIC aid may be cheaper (public domain), Power and PAL may have some advantages (instant subroutines, BASIC program debugging aids, more convenient assembling), and other packages may have a few more features, but the suitability of the sysedt environment for source code development and the overall good design of every part of the system make TSDS a very strong competitor.

TPUG CONTACTS

TPUG OFFICE
5300 Yonge Street
Toronto, Ontario
M2M 5R2 416–733–2933

Membership Info:

Regular Member (attends meetings)	$35.00 Cdn.
Student (full-time, attends meetings)	$25.00 Cdn.
Associate (Canada)	$25.00 Cdn.
Associate (U.S.)	$25.00 U.S.
Associate (Overseas – sea mail)	$35.00 U.S.
Associate (Overseas – air mail)	$45.00 U.S.

TPUG BBS (416) 273–6300

Board of Directors 1986/87

President	Anne E. Gudz	c/o 416/733–2933
Vice–President	Meyer Toole	416/229–2700
Vice–President	Carl Epstein	416/492–0222
Secretary	John Easton	416/251–1511
	David Bradley	c/o 416/733–2933
	Richard Bradley	c/o 416/733–2933
	George Davis	c/o 416/733–2933
	Mike Donegan (evgs.)	416/639–0329
	Keith Falkner	416/481–0678
	J. Allan Farquharson	c/o 416/733–2933
	Gerry Gold	416/225–8760

Meeting Coordinators

C–64 Chapter	Keith Falkner	416/481–0678
	Gord Campbell	416/492–9518
COMAL	Donald Dalley	c/o 416/733–2933
	Victor Gough	416/677–8840
Eastside Chapter	Don Farrow	c/o 416/733–2933
	Jim Hamilton	c/o 416/733–2933
Westside Chapter	John Easton	416/251–1511
	Al Farquharson	519/442–7000
Brampton Chapter	William Barrett	c/o 416/733–2933
68000 (nee SuperPET)	Gerry Gold	416/225–8760
	Avy Moise	c/o 416/733–2933
VIC 20	Anne E. Gudz (evgs.)	416/766–9307
C128	George Skinner	c/o 416/733–2933
Amiga Chapter	Mike Donegan (evgs.)	416/639–0329

Librarians

COMAL	Victor Gough	416/677–8840
PET	Mike Donegan (evgs.)	416/639–0329
SuperPet	Bill Dutfield	416/224–0642
VIC 20	Richard Best	c/o 416/733–2933
Commodore 64	George Davis	c/o 416/733–2933
C16–Plus/4	Paul Kreppenhofer	c/o 416/733–2933
B-128	Paul Aitchison	c/o 416/733–2933
Amiga	Mike Donegan (evgs.)	416/639–0329
Commodore 128	Adam Herst	c/o 416/733–2933
MS/DOS	Colin Justason	c/o 416/733–2933

TPUG Bulletin Board

Sysop (voice, weekdays)	Sylvia Gallus	c/o 416/896–1446
Assistant Sysop	Steve Punter	c/o 416/896–1446

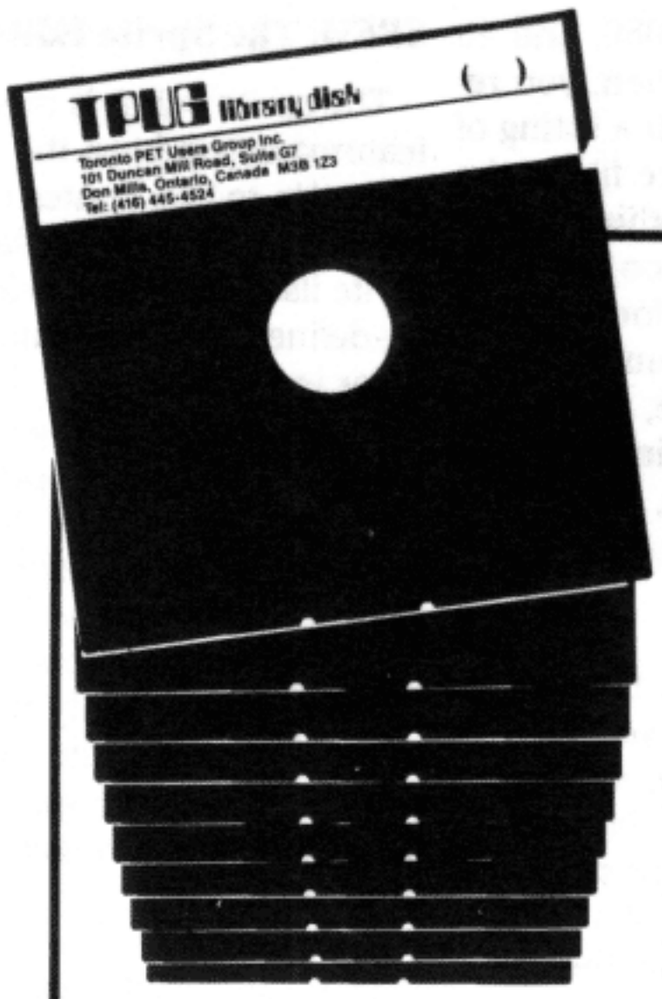
TPUG Unclassifieds

Desperately seeking Administrator Software (8032/8050) by Professional Software Inc. Call collect Jim Shurter (402) 399-8282.

Wanted: SuperPEts and 8050 disk drives. Call Scott (416) 625-0930 (Bus.)

For Sale: 8096 Computer, 8250 disk drive and miscellaneous software. Call Lynn (705) 748-4803.

Wanted: people to phone free BBS. There will be a section in the Commodore section for reviews and software lists for the Commodore, but we need enough callers, so leave word to Stuart Fell and it will be set up. Call (204) 687-8665. 24 hours per day.



TPUG LIBRARY DISKS

10 months of the latest, fabulous public domain software only

$79.95 CANADIAN
US orders US$59.95

Subscribe now to start receiving the TPUG disk of the month

You **SAVE 20%** or more by using this offer as a TPUG member

THIS OFFER VALID FOR PAID UP TPUG MEMBERS ONLY
PLEASE PRINT

Name: _____ Membership # _____

Address: _____

Type of Payment

☐ Cheque ☐ Mastercard ☐ Visa ☐ Money Order
Amount $ _____
Currency ☐ Can. ☐ US
Credit Card # _____
Expiry Date _____

Signature _____

Type of Computer

☐ C 64
☐ VIC 20
☐ PET ☐ 4040 ☐ 8050
☐ SUPERPET (10 disks) ☐ 4040 ☐ 8050
☐ C128 (1541 only)
☐ MS/DOS
☐ AMIGA (3½") (7 disks)

Software order form

NAME _____
STREET ADDRESS _____
CITY/TOWN/P.O. _____
PROV/STATE _____ POSTAL/ZIP CODE _____
TELEPHONE _____ MEMBERSHIP NO. _____

TORONTO PET USERS GROUP, 5300 Yonge Street, Toronto, Ontario, M2M 5R2 416–733–2933

disks

To order club disks by mail, send $10.00 for each 4040/2031/1540/1541 disk (4040 format), discount price 5-10 $9.00 each, 11 or more $8.00 each; and $12.00 for each 8050/8250 disk (8050 format). We do honour purchase orders from school boards.

These disks are for use with a _____ computer and a _____ disk drive.
Please send me the following:

3 Letter/No. Code	Description	4040 or 8050 Format	Price

Total $ _____.00

tapes

To order VIC 20 or Commodore 64 library tapes, send $6.00 for each tape.
To order PET/CBM or Commodore Educational Software, send $10.00 for each tape.

These tapes are for use with a _____ computer and a datasette.
If for a PET computer, what model - _____ - BASIC - 1.0[ ]; 2.0[ ]; 4.0[ ]?

3 Letter/No. Code	Description	Price

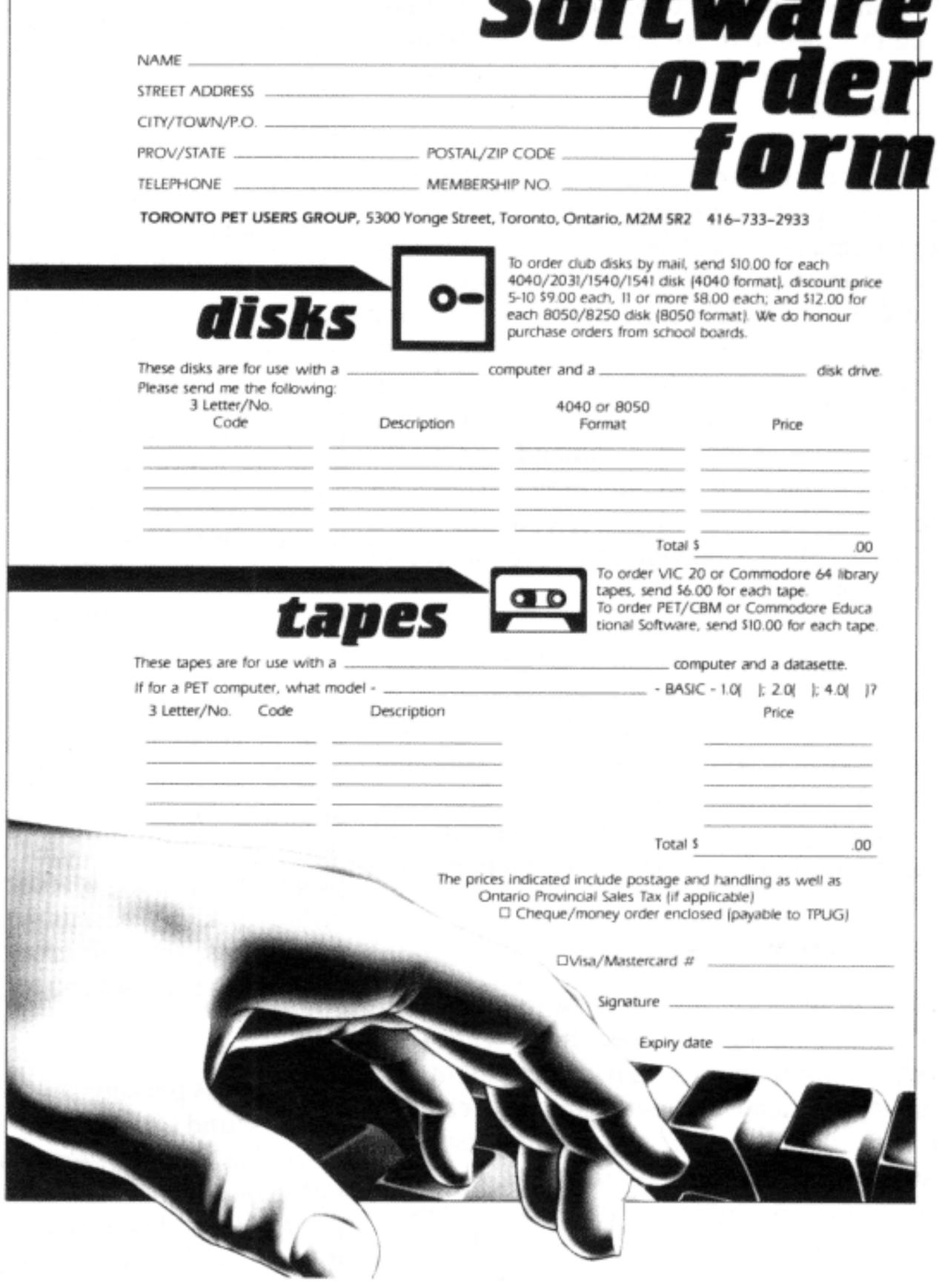
Total $ _____.00

The prices indicated include postage and handling as well as Ontario Provincial Sales Tax (if applicable)
☐ Cheque/money order enclosed (payable to TPUG)

☐Visa/Mastercard # _____

Signature _____

Expiry date _____



TPUG AMIGA PUBLIC DOMAIN DISKS

This public domain software is made available to you, the AMIGA owner, by the Toronto PET Users Group (TPUG). The disks were put together by Fred Fish of California.

Documentation for each disk is available in the README.list file.

We hope you enjoy these disks. Happy Computing

Mike Donegan – Amiga Librarian
(416)639–0329 evenings 8–11 PM (Toronto time)

README.listAAA

This is disk 1 of the freely distributable AMIGA software library.

amigademo Graphical benchmark for comparing amigas. Author: Gary Girzon

amigaterm Terminal emulation program with xmodem upload/download capability. Author: Michael Mounier

balls Simulation of the ''kinetic thingy'' with balls on strings where only the end balls move (quick, can YOU come up with a better description?). Anyway, cute. Author: Perry Kivolowitz

colorful Shows off use of hold–and–modify mode. Posted to usenet by Robert Pariseau.

dhrystone Dhrystone benchmark program. Author: Reinhold Weicker (Ada version), Rick Richardson (C version)

dotty Source to the ''dotty window'' demo on the Workbench disk. Posted to usenet by Dale Luck.

freedraw A small ''paint'' type program. Free drawing, boxes, filled boxes, etc. Author: Rick Ross

gad ''Fun with Gadgets''. Demonstration program for use of gadgets. Author: John Draper (Aka ''crunch'')

gfxmem Graphical memory usage display program. Watch your machine's memory usage! Cute and useful. Author: Louis Mamakos

halfbrite Sample program that demonstrates ''Extra–Half–Brite'' mode on latter AMIGA's with new VLSI chip. Allows 64 colors in low–res mode, rather than 32. Posted to usenet by Robert Pariseau.

hello Demonstrates creation of a simple window, ''hello world''. Posted to usenet by Eric Lavitsky.

latffp Shows how to access the Motorola Fast Floating Point library from Lattice C. Also demonstrates the tremendous speedup obtained. Author: Larry Hildenbrand

palette Sample program for designing color palettes. Author: Charlie Heath

trackdisk Demonstrates use of the trackdisk driver. Useful example of ''raw'' disk read/write. Author: Rob Peck

requesters Sample program and documentation for building and using requesters. John worked REAL hard to dig out all the information in this one! Author: John Draper (aka ''crunch'')

speech Sample speech demo program. Stripped down version of ''speechtoy''. Author: Rob Peck

speechtoy Another speech demo program. Cute. You have to see this one. Be sure to click gadget that pops up the face. Author: David Lucas

README.listAAB

This is disk 2 of the freely distributable AMIGA software library.

alib Object module librarian. Author: Mike Schwartz

cc Unix–like frontend for Lattice C compiler. Author: Fred Fish

dbug Macro based C debugging package. Machine independent. Provides function trace, selective printing of internal state information, and more. Author: Fred Fish

make Subset of ''unix'' make command. Useful, but does not have many of the features of the full make, much less the newer ''augmented make''. Author: Landon Dyer

make2 Another make subset command. Author: Marc Mengel

microemacs Small, relatively portable version of emacs. Has keyboard macros. No extension language. Author: Dave Conroy

portar Portable archiver. Used to bundle text file up into a single file for transmission as a unit, or otherwise handling as a single file. ''Portable'' because the code itself is portable and because the archive format is very simple (uses ascii headers to separate files). From Decus C distribution. Author: Martin Minow

xrf C cross reference utility. Originally from Decus C distribution. Author: Bob Denny

README.listAAC

This is disk 3 of the freely distributable AMIGA software library.

gothic Gothic banner printer. Prints DOWN the page, rather than across, so arbitrarily long banners can be created. Send EOF (CTRL–\) to end input. From a Decus C distribution several years ago. Author: unknown

roff A ''roff'' type text formatter, roughly following ''Software Tools'' version. Somewhat upwardly compatible with unix ''nroff'' command. Author Ken Yap

ff A very fast text formatter, controlled exclusively by command line arguments. Author: Gary Perlman & hordes of students

cforth A highly portable forth implementation. Lots of goodies. Author: Allan Pratt

xlisp A nice little lisp implementation. Compiles and links ok, but something in the Lattice C setjmp/longjmp code prevents it from currently running. Might be easily fixed. Version 1.4. Author: David Betz

README.listAAD

This is disk 4 of the freely distributable AMIGA software library.

banner Prints horizontal banner (across screen). From Decus C distribution of several years ago. Author: Unknown

bgrep Another grep like utility, also using the Boyer–Moore algorithm. Author: Roy Mongiovi and Arnold Robbins

bison A replacement for unix ''yacc'' command. This is from the GNU (GNU is Not Unix) effort, and was obtained from the Free Software Foundation. Compiles and links (with some effort) but currently crashes the machine. Needs work, but will probably be worth it. Author: Bob Corbett and Richard Stallman

bm A grep like utility using the Boyer–Moore algorithm. Author: Peter Bain

grep Decus grep (Get Regular Expression and Print). Useful for finding strings in files. Author: Unknown

kermit This is an absolutely ancient kermit, who's only saving grace is that it is small and quite portable. On the AMIGA, there is no connect mode, only send and receive. You must log into the remote machine via one of it's local terminals and point it's kermit at the appropriate serial line connected to the AMIGA. Author: Unknown, but it is so hacked up it doesn't matter by now.

MyCLI Another CLI for the AMIGA. Author: Mike Schwartz

mandel A Mandelbrot set program. Author: Robert French, with additions by RJ Mical

README.listAAE

This is disk 5 of the freely distributable AMIGA software library. It contains many of the programs, written by Commodore–Amiga people, that were included in disk number 1. However, they have been cleaned up so that they compile cleanly on the AMIGA, return resources allocated, etc.

cons Console device demo program with supporting macro routines. Author: Rob Peck 12/1/85

freemap Creates a visual diagram of free memory. Comes with custom icon so can be run from workbench screen. Author: Robert J. Mical

input.dev In combination with hander.interface.asm (supplied), lets a user trap keyboard or mouse events before they get to Intuition and if desired, install new (phony/generated/other–devices–mouse–simulations) into the input stream. Handler.interface.asm is needed to convert the calling sequence performed by the input.task for the input stream management into something that a C program can understand. Author: Rob Peck 12/1/85

joystick Shows how to set up the gameport device as a joystick. Reports parameters received from joystick hooked to right port. Author: Rob Peck 12/1/85

keyboard Sample program to demonstrate direct communications with the keyboard. Author: Rob Peck 12/1/85

layers Shows use of the layers library, used by Intuition to create windows (layers handles all overlapping drawing areas and keeps things straight, sending to the back, bringing to the front, making larger/smaller, etc.) Author: Rob Peck 12/1/85

mandelbrot Latest and greatest mandelbrot program. New features include the ability to save images in ''iff'' format, for reading into Deluxe Paint or other programs that use iff. Also includes code from Graphicraft to handle color palette and change colors at will. Many additions by RJ Mical. Author: Robert French

mouse Shows how to set up the gameport device as a mouse so that hooking up the mouse to the right port gives access to mouse information. Author: Rob Peck 12/1/85

one.window Produces a window with a console attached. Does graphics in the top half and limits the console activity to the lower half. Author: Rob Peck 12/18/85

parallel Demonstrates access to the parallel port. Author: Tom Pohorsky 12/1/85

printer Shows how to open and use a printer, does a screen dump of the workbench screen if there is a graphics–capable printer attached. Currently does not compile under Lattice C Ver3.02 and does not link under Lattice C Ver3.03. Author: Rob Peck 12/1/85

print.support Printer support routines. Currently won't compile, missing a header file ''local.h''. Author: unknown

proctest Sample code to create a process, set up message ports, pass messages, etc. Sample slave code for create process test (littleproc.c) currently does not link under either Lattice C 3.02 or 3.03 (unresolved global variables not in libraries). Author: Rob Peck 1/4/86

region Demonstrates how a drawing area can be split into linked rectangular regions. Draws a rectangle in a single playfield display, then draws ''Behind a Fence'' several times behind an apparent fence in the rectangle. Only works under Lattice Ver3.03. Author: Rob Peck 12/1/85

samplefont A sample font that produces clubs, hearts, spaces, and diamonds as its four characters. Shows precisely what is contained in an Amiga font. Author: Unknown

serial Demonstrates access to the serial port. Requires an external terminal connected to the serial port. Author: Unknown

singlePlayfield Creates and displays a 320 by 200 by 2 bit plane single playfield display. Completely covers Intuition's display, but gives the system back at exit. Author: Rob Peck 12/1/85

speechtoy Latest and greatest version of Dave's cute speech demo program. Comes with custom icon so can be run from a workbench screen. Author: Dave Lucas

speech.demo A much simplified version of speechtoy. Also includes exec support functions for extended IO requests, CreateExtIO() to allocate and initialize a new IO request block and DeleteExtIO() to free an extended IO request block. Author: Rob Peck 12/1/85

text.demo Sample program that asks AvailFonts() to make a list of the fonts that are available, then opens a window and then prints a description of the various attributes that can be applied to the fonts, in the font itself. Previous versions were released as ''whichfont''. Author: Rob Peck 12/1/85

timer Simple timer example program. Includes dynamic allocation of data structures needed to communicate with the timer device, as well as the actual device IO. Author: Rob Peck 12/1/85

trackdisk Demonstrates use of trackdisk driver. Useful example of ''raw'' disk read/write. Author Rob Peck 12/1/85

README.listAAF

This is disk 6 of the freely distributable AMIGA software library.

compress Compress reduces the size of the named files using adaptive Lempel–Ziv coding. The amount of compression obtained depends on the size of the input, the number of bits per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50–60%. Compression is generally much better then that achieved by Huffman coding (as used in ''pack'') or adaptive Huffman coding (as used in ''compact''), and takes less time to compute. Great for saving disk space, or reducing transmission time over phone lines! Also includes ''btoa'' and ''atob'' for converting binary files to ascii, and then back to binary (for transmission over links that do not support 8 bits). Author: Thomas Spencer, with improvements by others

dadc A digital computer impersonating an analog clock impersonating a digital clock. Author: Perry S. Kivolowitz

microemacs An upgraded version of microemacs originally distributed on disk number 2. I don't have an exact list of changes but the new executable is about 25% larger, so there must be something in there! One enhancement is that some functions are now also bound to function keys. Author: Dave Conroy, with enhancements by Jack Roose.

mult Mult reads the input comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are output, and the remainder of the lines are removed. Repeated lines must be adjacent in order to be found. Options are present to output the first of multiple lines, for comparing adjacent lines by field only, and for specifying the field separator character. Kind of the opposite of the Unix ''uniq'' program. Author: Dennis Bednar

scales Demonstrates use of the Audio functions in the ROM to produce four voice sound. It uses a simple waveform (sawtooth) with no amplitude control (ie, envelope) or frequency variation (ie, vibrato), but these can be easily implemented. Also includes considerable documentation on audio device. Author: Steven A. Bennett

setparallel Allows the CLI user to dynamically change any particular parallel port parameter. Author: Keith Stobie and Tom Pohorsky

setserial Allows the CLI user to dynamically change any particular serial port parameter. Author: Keith Stobie and Tom Pohorsky

sortc A quicksort based sort program, using separate driver and sort modules. Originally from a DECUS C distribution. Claims to be fast, but when operating entirely out of ram takes 93 seconds to sort its input file, sort.c, while AmigaDos's sort takes only 43 seconds under the same conditions. Author: Dave Conroy, Martin Minow, and Ray Van Tassle

stripc Strips comments and extraneous whitespace from C source files. Useful for compacting the C header files to increase usable disk space. Author: Chris Metcalf

README.listAAG

This is disk 7 of the freely distributable AMIGA software library. This disk contains a port (executables only) of the popular UNIX game ''Hack'', courtesy of John Toebes, 120 H Northington Pl, Cary NC. 27511. This is Version 1.0.1.

README.listAAH

This is disk 8 of the freely distributable AMIGA software library. It contains the source to ''Hack'', a popular UNIX game ported to the AMIGA by John Toebes. Please read all the associated README files before redistributing with changes.

README.listAAI

This is disk 9 of the freely distributable AMIGA software library.

moire	Draws moire patterns in black and white in a borderless backdrop window. Currently only runs with Lattice C version 3.02, when compiled with 3.03 crashes the system. Author: Scott Ballantyne
MVP-FORTH	Mountain View Press Forth, version 1.00.03A. A shareware version of forth from Fantasia Systems. Runable from the workbench, comes with icons and such. If you use this, you should send a contribution to Fantasia Systems / Mountain View Press.
proff	Another text formatting program. This one is significantly more powerful than any of the others previously distributed on these disks.
setlace	Program to toggle interlace mode on and off. When used with a black background, and amber or green characters, produces a nice CLI environment. Author: Bob Pariseau, Nov 85
skewb	Not quite sure what this is supposed to be, it was downloaded from a bbs with no documentation. C'mon folks, at least put a three line description at the start of your programs! Author: Raymond S. Brand
sparks	Graphics demo that draws a ''moving pathway'', adding to the front and removing from the tail (sure is hard to come up with verbal descriptions of these things!). Author: Scott Ballantyne

README.listAAJ

This is disk 10 of the freely distributable AMIGA software library.

conquest	You control an interstellar empire, decide which star systems to explore, which planets to colonize, etc. The computer will also be building its own empire and competing with you for resources. The one with the greatest population at the end wins. Executable only, no source. Downloaded from Denver BBS. Author: Unknown, ported to Amiga by Rob Shimbo
dehex	Simple little program to convert hex format file into a binary file. Reads lines of hex characters on stdin and writes 8 bit dehex'd bytes to stdout. Author: Fred Fish
filezap	Patch program for any type of file. Executable only, no source available. Author: John Hodgson
fixobj	Strip extraneous garbage off end of object files transfered with xmodem. Does not require preknowledge of actual file length, uses knowledge of Amiga file structure. Author: John Hodgson
iff	Routines to read and write iff format files. Includes program to display iff files made by DPaint. Also see the DPSlide program on disk 11. Author: Jerry Morrison, Steve Shaw, and Steve Hayes
ld	I presume this stands for ''list directory''. Lists contents of specified diretory or disk, using inverse video and colors to group objects by type. Author: Dave Haynie
ls	Minimal implementation of UNIX ls. Demonstrates use of UNIX style filename pattern matching code. Author: Rick Schaeffer
sq.usq	Squeeze and unsqueeze, from the CPM world. Works, but ''compress'' from disk 6 runs almost twice as fast and produces even smaller files. Author: R. Greenlaw, Amiga port by Rick Schaeffer
trek73	A Star Trek game. This one is just crying out to be Amiga'ized by some ambitious hacker. Authors: Many, see comment in main.c
yachtc	Dice game. Author: Sheldon Leemon

README.listAAK

This is disk 11 of the freely distributable AMIGA software library. This is a picture disk.

README.listAAL

This is disk 12 of the freely distributable AMIGA software library. Note that this disk contains a number of programs apparently available only in executable form. I am not real happy with this trend, but had no way of getting in contact with the authors who posted these programs to various BBS's to request source. I have no reason to believe that any of these executable only postings contain any deliberate ''misfeatures'', but you should always use such programs cautiously.

amiga3d	Shows a rotating 3 dimensional solid ''Amiga sign''. No source available, executable only (downloaded from a Denver BBS) Author: Barry (bart) Whitebook of C-A
ArgoTerm	Yet another terminal emulator program. However, this one is written in macro assembler, so should be of interest to anyone doing serious assembly language development. Version 0.20 Author: Jez San
arrow3d	Shows a rotating 3 dimensional wire frame arrow. No source available, executable only (downloaded from a Denver BBS) Author: Steve Beats of C-A
ld4	Another directory listing program using some graphics style output. Executable only, no source available. (Downloaded from a Denver BBS) Author: unknown
IconExec	These two tools allow execution of a program from a SetWindow icon without having to recompile the program. Author: John Toebes VIII
images	Miscellaneous DPaint and digitized pictures, in iff format. Some are rated R.
SetAlternate	Merge the images from two icons to produce one icon with a primary image, and a possibly completely different image to display when selected. Author: John Toebes VIII
StarTerm	Another terminal emulator program. ASCII and XMODEM support, telephone dialer, function keys, load file stripping, text file conversion, full/half duplex, Author: Jim Nangano and Steve Plegge

README.listAAM

This is disk 13 of the freely distributable AMIGA software library.
It is a grab bag of basic programs, mostly untested. No speekee basic so if these don't work, call someone else.

README.listAAN

This is disk 14 of the freely distributable AMIGA software library. It contains the first ever public release of two new pieces of code, ''dex'' and ''termcap''. Also, this is the first disk in the library that includes executables produced with both the Manx and Lattice C compilers, whenever possible. This greatly helps to isolate bugs.

amiga3d	Shows a rotating 3 dimensional solid ''AMIGA'' sign. This is an updated version of the program released on disk number 12, and now includes full source. Author: Barry Whitebook @ Commodore-Amiga
beep	Source for a function that generates a beep sound, like CTRL-G on a VT100 terminal. Author: Samuel Dicker @ Commodore-Amiga
dex	Program to extract documentation in a human readable format inside source files, and produce nroff style output for manuals and other such external documents. First ever public release. Author: Fred Fish
dimensions	Programs to demonstrate three and four dimensional graphics. Not quite sure how else to describe them! Author: Anselm Hook
filezap	An updated version of the file zap utility first released on disk number 10. Can be used to patch any type of file. Nice, and VERY useful. Author: John Hodgson
gfxmem	An updated version of the graphical memory display program first released on disk number 1. Watch your machine's memory usage change dynamically under use! Author: Louis Mamakos
gi	Converts DPaint brush files to C source files ''necessary to create an Image structure, including height, width, depth, and color information, as well as the array of data which represents the bit planes of the image''. Author: Mike Farren
pdterm	A simple terminal emulator that does ANSI or DEC VT-100 emulation in 80 cols by 25 lines. Version 1.21. Author: Michael McInerny
shell	A simple csh style shell with history and some other goodies. Still needs some polishing and enhancement, but is quite nice as it. Thanks Matt!!! We've really needed something like this for a long time. Now if you would just do a ksh version instead. . . Author: Matt Dillon
termcap	A (mostly) unix compatible implementation of a termcap library. First ever public release. Author: Fred Fish

README.listAAO

This is disk 15 of the freely distributable AMIGA software library. Whenever possible, both Lattice C and Manx C executables are provided. The Lattice executables have the extension ''.lattice'' and the Manx executables have the extension ''.manx''. This greatly helps to isolate bugs.

Blobs	A simple graphics program, reminiscent of the unix ''worms'' program, but in color of course. Author: Peter Engelbrite
Clock	A simple digital clock program designed to be small and to live completely in the screen title bar, where it is out of the way. Author: Mike Meyer
Dazzle	An eight-fold symmetry dazzler program. Really pretty! Author: Peter Engelbrite
Fish	''A demo program which runs an AnimOb in a double buffered screen with sequence cycled animation''. (Basically shows a fish ''swimming'' across the top of the screen). Author: Catherine Wagner (posted by Barry Whitebook)
Monopoly	A really nice monopoly game written in AbasiC. Follow the directions in the file ''InstallationGuide'' to produce a bootable games disk. Author: David Addison
OkidataDump	Okidata ML92 driver and WorkBench screen dump program. Does both alpha and graphics. Untested (I don't have the printer) Author: Raimund Gluecker
Polydraw	A drawing program written in AbasiC. Author: David Addison
Polyfractals	A fractal program written in AbasiC. Author: David Addison

README.listAAP

This is disk 16 of the freely distributable AMIGA software library. It is a copy of the Amiga Developer's IFF disk, received directly from Commodore-Amiga sources, with permission to place in the library and redistribute. It is unchanged except for:

1) The volume name has been changed to AmigaLibDisk16.
2) This file (README.list16) has been added.
3) The standard library README file README.dist has been added.

Enjoy -Fred

README.listAAP

This is disk 16 of the freely distributable AMIGA software library. This is a copy of the Amiga Developer's IFF disk, received directly from Commodore-Amiga sources, with permission to place in the library and redistribute. It is unchanged except for:

1) The volume name has been changed to AmigaLibDisk16.
2) This file (README.list16) has been added.
3) The standard library README file README.dist has been added.

Enjoy -Fred

README.listAAQ

This is disk 17 of the freely distributable AMIGA software library. This is a copy of a H.A.M. (Hold And Modify) graphics demo disk received from a vendor that is producing hardware to capture such images, and software to process them. It is unchanged except for:

1) The volume name has been changed to AmigaLibDisk17.
2) This file (README.list17) has been added.
3) The standard library README file README.dist has been added.
4) The 'showilbm' program has been added to allow viewing of the two monochrome pictures (dozer.hires and robert.lores) without requiring Deluxe Paint.
5) The supplied 'readme' file has been augmented with information received on hardcopy along with the disk, and a note has been added about how to use the showilbm program.

Enjoy -Fred

README.listAAR

This is disk 18 of the freely distributable AMIGA software library.

AmigaDisplay	Yet another variation of a terminal emulator program. This is a modified AmigaTerm that can emulate a dumb terminal (interesting paradox here somewhere. . .), translate line termination sequences, optionally capture or discard control characters in the captured file, use audible bell, use another font, etc. Author: Don Woods (Original code by Michael Mounier)
Ash	Prerelease version of a C-shell like shell program. Has history, command substitution, loops, etc. Author: Thorn Smith
Browser	A program that lets you wander around a file tree and peek into files, all with the mouse. Author: Mike (I'll be mellow when I'm dead) Meyer
MC68010	Complete information package for upgrading an Amiga to use an MC68010 in place of the MC68000. Includes a software fix that makes this transparent to user programs that use instructions that are priviledged on the 68010. Author: Thad Floryan
Multidim	Lets you rotate a 2 to 6 dimensional ''cube'' on the screen using the joystick. Author: Robert French
PigLatin	Tired of the ''say'' command? This one will translate and speak your input in pig latin! Author: Thomas Clement
Scrimper	Short for ''SCReen IMage PrintER''. A screen dump utility which can be run from the workbench or the CLI. Author: Perry Kivolowitz
Xlisp1.6	A very nice little lisp for those that want to study the internals of a real, working lisp interpreter. Author: Dave Betz

README.listAAS

This is disk 19 of the freely distributable AMIGA software library.

BlackJack	A line oriented (no graphics) blackjack game. Author: Unknown
JayMinerSlides	These are the Amiga slides produced/used by Jay

Miner (the designer of Amiga's custom graphics chips) in his talks about the Amiga. They are all hi-res (640 x 400) and are best displayed on a long-persistence color monitor.

Keymap_Test A program to test the keymapping routines and find possible bugs. Useful as an example of keymapping. Author: Pushpa Kumar

LockMon Find file locks. Useful for discovering if programs properly clean up after themselves. Author: Dewi Williams

README.listAAT

This is disk 20 of the freely distributable AMIGA software library.

AmigaToAtari Source code for an Amiga to Atari ST object code format converter. Takes Amiga objects as input and produces Atari objects as output. This allows the Amiga to be used as a cross development machine providing the proper libraries are available. (Currently does not work, but I don't have an ST to try it with anyway. . .) Author: Landon Dyer at Atari Corp.

DiskSalv Program to recover files from a trashed AmigaDOS disk. Can also ''undelete'' files deleted by mistake, so long as they have not overwritten by further disk activity. Requires two disk drives. VERY useful. . . Author: Dave Haynie

Hash Small example program that computes the AmigaDOS directory hash function. Author: Neil Katin at Commodore-Amiga

Hd Hex dump utility using some ideas from Mike Higgin's article in Computer Language magazine, Apr 86. Formats the dump based on the natural byte ordering of the machine on which it runs. Author: David Elins at NEC Information Systems

MandelBrots Some mandelbrot images submitted for the ''mandelbrot images contest'' some months ago. Only three people submitted mandelbrots and these were among the most interesting.

MultiTasking Tutorial and example program for multitasking at the Exec level. Author: Leo L. Schwab

Pack Program to strip extraneous whitespace from C programs or header files. Can be used to condense the C compiler header files to free up disk space. Author: Jeff Dickson

PortHandler A sample Port-Handler program that performs the functions of the standard Port-Handler. Shows what the BCPL environment looks like from the handler point of view. Author: John Toebus VIII

Random Random number generator in assembly. Much faster than versions using floating point. Can be used by either assembly or C programs. Author: Steve Beats at Commodore-Amiga

SetMouse2 Program to set the mouse port to either the left port or right port. Author: Robert Burns at Commodore-Amiga

SpeechTerm Terminal emulator that can speak the received text. Also has XMODEM file transfer. Author: Leftheris Koutsofios

Ted Demo version of an editor that has since been renamed as TXed (I believe). Author: Charles Heath at MicroSmiths

README.listAAU

This is disk 21 of the freely distributable AMIGA software library. This is a copy of Thomas Wilcox's Mandelbrot Set Explorer disk. It is unchanged except for:

1) The volume name has been changed to AmigaLibDisk21.
2) This file (README.list21) has been added.
3) The standard library README file README.dist has been added.

To run from CLI:

1) cd dfX: (where X is drive containing disk)
2) mse

To run from Workbench:

1) Click on MSD icon.

It contains extensive on-line help information, unlike other Mandelbrot programs distributed in this library. It also is capable of displaying some very pretty hi-res pictures in interlace mode.

Enjoy -Fred

README.listAAV

This is disk 22 of the freely distributable AMIGA software library. This disk contains two new ''strains'' of microemacs, both derived from early releases of Dave Conroy's microemacs. There is currently an attempt on usenet, lead by Dave Brower, to coordinate an effort to merge features from the different versions into a single supported microemacs. In the meantime, perhaps you can find a feature you need in one of these. . .

Lemacs Microemacs version 3.6 as enhanced and supported by Daniel Lawrence. This version works and has been tested on Unix V7, BSD 4.2, Amiga, MS-DOS, and VMS. Enhancements include overwrite mode, support for Amiga function keys, reverse video status line, numeric arguments using <ESC><number>, replace, buffer specific editing modes, word wrap mode, goto-line, buffer rename, insert-file, execute named command, describe bindings, startup files, and more. Author: Dave Conroy, enhancements by Daniel Lawrence

Pemacs Microemacs as enhanced by Andy Poggio. New features include use of <ALT> keys as Meta keys, default buffers on buffer switch, mouse support, higher priority to improve interactive response, creation of backup files, paragraph fill, word wrap, query replace, and support for function keys. Author: Dave Conroy, enhancements by Daniel Lawrence

README.listAAW

This is disk 23 of the freely distributable AMIGA software library. It contains a significantly enhanced version of microemacs based on the version 30 release posted to usenet's mod.sources newsgroup.

Previous versions of microemacs released on these disks derived from a very old version of microemacs. Since that old release, the author has cleaned up lots of loose ends and restructured major parts of the code. Other people have already added support for termcap, ports to other machines and operating systems, and a limited GNU emacs compatibility option.

Because recipients of this disk might wish to run this new microemacs on other machines in addition to the Amiga, I have encluded all sources that were available to me at the time of release. The subdirectories which give alternate implementations are:

Sys/Vms	System modules for VAX VMS
Sys/Atari	System modules for Atari ST
Sys/Msdos	System modules for MSDOS
Sys/Cpm86	System modules for Cpm
Sys/Amiga	System modules for Amiga
Sys/Ultrix	System modules for Ultrix/BSD4.2
Sys/SysV	System modules for Unix System V
Tty/Atari	Terminal driver for Atari ST
Tty/7300	Terminal driver for AT&T Unix PC (7300)
Tty/Intuition	Terminal driver for Amiga using Intuition
Tty/Termcap	Terminal driver using termcap for Unix
Tty/Ansi	Terminal driver using Ansi codes
Tty/Amigados	Terminal driver for Amiga using just Amigados
Tty/Heath	Terminal driver for a Heath terminal

There currently is a major effort underway on usenet, led by Dave Brower, to standardize microemacs and bring the many variations under one common implementation. Please send any enhancements to rtech!daveb, or send them to me and I will see that he gets them.

README.listAAX

This is disk 24 of the freely distributable AMIGA software library.

Conquest You control an interstellar empire, decide which star systems to explore, which planets to colonize, etc. The computer will also be building its own empire and competing with you for resources. The one with the greatest population at the end wins. First distributed in executable form only on disk number 10. This distribution includes source. Author: Unknown, ported to Amiga by Rob Shimbo

Csh Second release of a csh-like alternative to the CLI, first released on disk number 14. Has alias, builtin functions like ''dir'' for speed, history, named variables, command re-execution with substitution, etc. Author: Matt Dillon

Modula-2 A pre-release version of the single pass Modula-2 compiler originally developed for MacIntosh at ETHZ. This code was transmitted to the AMIGA and is executed on the AMIGA using a special loader. Binary only.

README.listAAY

This is disk 25 of the freely distributable AMIGA software library. This disk contains a port (executables only) of the popular UNIX game ''Hack'', courtesy of John Toebes, 120 H Northington Pl, Cary NC. 27511. This is Version 1.0.1E., with graphics enhancements for the Amiga.

README.listAAZ

This is disk 26 of the freely distributable AMIGA software library.

UnHunk Tool to process the Amiga ''hunk'' loadfile format. Collects code, data, and bss hunks together, allows individual specification of code, data, and bss origins, and generates binary file with format reminiscent of Unix ''a.out'' format. The output file can be easily processed by a separate program to produce Motorola ''S-records'' suitable for downloading to PROM programmer. Author: Eric Black

C-kermit Port of the popular ''kermit'' program, a flexible virtual terminal and file transfer program from Columbia University. Author: Unknown, ported to Amiga by Davide Cervone

Ps A program to display process priorities and another program to set a process's priority to any arbitrary valid value. Very useful to provide finer control over your multitasking environment. Author: Dewi Williams

Archx Yet more programs for bundling up text files and mailing or posting them as a single file unit. Author: Martin Minow

README.listABA

This is disk 27 of the freely distributable AMIGA software library.

ABdemos Amiga Basic demos. NewConvertFD creates .bmaps from fd files. BitPlanes finds addresses of and writes to bitplanes of the screen's bitmap. AboutBmaps is a tutorial on creation and use of bmaps. LoadILBM loads and displays IFF ILBM pics. LoadACBM loads and displays ACBM pics. ScreenPrint creates a demo screen and dumps it to a graphic printer. Author: Carolyn Scheppner, Commodore-Amiga

Disassem A simple 68000 disassembler. Reads standard Amiga object files and disassembles the code sections. Data sections are dumped in hex. The actual disassember routines are set up to be callable from a user program so instructions in memory can be disassembled dynamically. Author: Bill Rogers, Univ of WAIKATO (New Zealand)

DvorakKeymap Example of a keymap structure for the Dvorak keyboard layout. Untested but included because assembly examples are few and far between. Author: Robert Burns @ Commodore-Amiga

Hypocycloids An electronic Spirograph inspired by an article in Feb '84 Byte. Author: Terry Gintz

LinesDemo A demo program which illustrates the use of proportional gadgets to scroll around in a superbitmap window. Posted to usenet by Barry Whitebook @ Commodore-Amiga

MemExpansion Schematics and directions for building your own homebrew 1 Mb memory expansion. Author: Michael Fellinger

SafeMalloc Ever have a program that corrupts it's own memory? Well, this little gem can help you find the problem in a hurry. Acts as an interface between your program and the real malloc, checking for overrun, underrun, and duplicate freeing of malloc'd space. Author: Bjorn Benson, modifications by Fred Fish

ScienceDemos Some science demos. Sidereal is a tutorial program which introduces the user to the relationships between Julian and calendar dates and solar and sidereal times. J2000 is a utility program which converts stellar positions, proper motion, parallax and radial velocity from the standard epoch B1950 (FK4) to epoch J2000 (FK5). Galilean is a tutorial program which determines the position of the Galilean satellites relative to Jupiter. Programs courtesy of David Eagle at Science Software.

README.listABB

This is disk 28 of the freely distributable AMIGA software library.

Backgammon Another game in AbasiC by the author of Monopoly. Author: David Addison

Cpp This is a copy of the Decus cpp, ported to the Amiga. This cpp is more powerful and complete than either of the built in cpp's in Manx or Lattice C. Also included is a modified version of the Unix like cc frontend, for Manx C, that knows about the cpp. Author: cpp by Martin Minow; cc by Fred Fish

Cribbage Another game in AbasiC by the author of Monopoly. Author: David Addison

MileStone Another game in AbasiC by the author of Monopoly. Author: David Addison

Othello Another game in AbasiC by the author of Monopoly. Author: David Addison

Shar A program which can pack and unpack archives compatible with the Unix ''shar'' (shell archiver) program. Author: Dave Wecker

SuperBitMap An example program that shows how to use ScrollLayer, how to sync the SuperBitMap prior to printing, and how to create a dummy rastport for dumping the SuperBitMap. Works under 1.2 and up. Author: Carolyn Scheppner and Phil Lindsay

README.listABC

This is disk 29 of the freely distributable AMIGA software library.

AegisDrawDemo Demo disk of the Aegis Draw program. This is the actual production program with only the ''save'' feature disabled and without documentation (intentionally). Very impressive program! Received directly from Aegis Development.

Cc Version of the Unix like C compiler frontend program, for Manx C. Previous released version was for Lattice C. Author: Fred Fish

Enough A new CLI execute file command that tests for various system resources such as available memory or existance of specific files, directories, or devices. Author: Bruce Barrett @ Commodore-Amiga

Player	Animation player for Aegis Animator. Received directly from Aegis Development.
Rubik	An animated Rubik's cube program based on Barry Whitebook's ''amiga3d'' program and Raymond Brand's ''skewb'' program. Author: Bill Kinnersley
StringLib	A public domain reimplementation of the Unix string library functions. Author: Henry Spencer @ U of Toronto
Vt100	A vt100 emulator program with Kermit and Xmodem file transfer protocols (based on AmigaTerm). Author: Dave Wecker

README.listABD

This is disk 30 of the freely distributable AMIGA software library. Note that the programs on this disk are all shareware/freeware. This means that if you find them useful, you are morally obligated to send a small donation to the author to help support his efforts and fund further development. I hope that by including them in this library I have furthered the author's goals of widest possible distribution and thus maximum return. Also note (for those that like to split hairs) that I am distributing this disk as a ''freebee''. To receive a copy, just request it when ordering any other disks from the library.

Bbs	A BBS for the Amiga (BBS–Amiga Version 1.1). Author: Ewan Grantham Author's suggested shareware donation: $25
FineArt	This is a collection of works from some of the best of current Amiga artists. Many thanks to Jim Sachs, Sheryl Knowles, Jack Haeger, and Aegis Development for submissions.
FontEditor	Does what the name implies, edits font's. Author: Tim Robinson. Author's suggested shareware donation: $5
MenuEditor	Create and edit menus, saving the result as either a binary file for further editing or as C source for inclusion in a program. Author: David Pehrson. Author's suggested shareware donation: $15
StarTerm	Starterm version 3.0. Very nice telecommunications program. Author: Jim Nangano. Author's suggested shareware donation: $20

README.listABE

This is disk 31 of the freely distributable AMIGA software library.

Life	Executes the cellular automata game of LIFE in the blitter chip. Uses a 318 by 188 display and runs at 19.8 generations per second. Author: Tomas Rokicki
Mandelbrot	Latest version of Robert French's mandelbrot program, version 3.00. Author: Robert French
MxExample	Example of mutual exclusion gadget handling. Author: Davide Cervone
RamSpeed	Program to measure raw memory speed, comparing internal memory to the external memory. Author: Perry Kivolowitz
Set	Replacement for the Manx ''set'' command (to set or change environment variables) with several improvements. Author: Tomas Rokicki
Tree	Draws a recursive tree (green leafy type, not files). Author: Robert French (original version by W. Balthrop)
TxEd	Latest demo version of Microsmith's text editor, TxEd. This is the full production version except that files are limited to 10K bytes in length and the search/replace functions are disabled. Also, demo is based on an older version of TxEd, new release has additional features. Author: Charlie Heath
VDraw	Drawing program, based on freedraw, but now transformed and enhanced beyond recognition. Draws hollow boxes, filled boxes, freehand sketches, lines, circles, and arcs. Also has area fill, text insertion, 16 different line patterns, predefined area fill patterns, a magnify mode, cut and paste, color inversion, erase, grid pattern, and more. This is version 1.08. Author: Stephen Vermeulen
Xicon	Contains two programs; Xicon which lets you use icons to call up scripts containing CLI commands, and Ticon, which is a simple program to display text files from icons. Author: Pete Goodeve

README.listABF

This is disk 32 of the freely distributable AMIGA software library.

Address	Extended address book written in AmigaBasic. Author: Mark Hurst
Calendar	Calendar/diary program written in AmigaBasic. Author: Mark Hurst
DosPlus1	First volume of CLI oriented tools for developers. Executables only. Author: Bill Beogelein
DosPlus2	Second volume of CLI oriented tools for developers. Executables only. Author: Bill Beogelein
MacView	Allows viewing of a standard MacPaint picture file in either Amiga low resolution (320 x 200) or high resolution (640 x 400). Executable only. Author: Scott Evernden
Puzzle	Simulation of puzzle with moving square tiles. Executable only. Author: Bill Beogelein
ShowHAM	Program to display HAM (Hold And Modify) mode images from the CLI environment. Executable only. Author: Unknown
Solitaire	Two new ABasiC games, Canfield and Klondike. Author: David Addison
Spin3	Simple program that creates spinning cubes and transforms them into op–art. Example of how to create a double buffered display with color tables that can be changed. Author: Ronald Peterson
Sword	Sword of Fallen Angel. Text adventure game written in AmigaBasic. Author: Andry Rachmat
Trails	Cute little program that leaves a trail behind the pointer when the mouse is moved. The trail has programmable symmetry, thickness, and length. The other interesting thing about this program is that it is written in Modula–2 (source provided). Author: Richie Bielak

README.listABG

This is disk 33 of the freely distributable AMIGA software library.

3dstars	3d version of Leo's ''stars'' program (also on this disk). Author: Leo Schwab
Bigmap	Program which demonstrates how to use the low–level graphics calls, ScrollVPort especially. Demonstrates scrolling around inside a very big bitmap. Author: Leo Schwab
Dbuf.gels	Sample program that demonstrates the animation routines for Bobs and VSprites. Uses double buffering to smooth the display motion. Author: David Lucas
DiskMapper	Displays sector allocation of floppy disks. Author: Leo Schwab
MemView	Program which sets up a direct window into RAM, thus dynamically displaying the contents of memory. Author: Leo Schwab
Oing	Displays a window full of little bouncing balls. Author: Leo Schwab
ScreenDump	Dumps rastport of highest screen/window to printer. Author: Carolyn Scheppner
Sdb	Simple database program, originally released on a DECUS VAX SIG tape. Author: Unknown, Amiga port by Mic Kaczmarczik
Sproing	Same as Oing but includes sounds of balls colliding with boundries. Author: Leo Schwab
Stars	Displays a screen full of stars, reminiscent of a view from the starship Enterprise's flight deck. Author: Leo Schwab
TermPlus	Yet another variant of Michael Mournier's AmigaTerm program. This one includes improved ascii capture, CRC and checksum xmodem protocol transfers, CompuServe B–protocol transfers, a phone library, function key support, and limited AmigaDOS functions. Author: Enhancements by Bob Rakosky
Vt100	Release 2.0 of Dave's version of AmigaTerm. Includes support for function keys and script files. Author: Enhancements by Dave Wecker and Steve Drew.

README.listABH

This is disk 34 of the freely distributable AMIGA software library.

Alint	Support files for Gimpel lint to make it useful on the Amiga. Author: Thomas Althoff
Blink	A linker written as a replacement for Alink. Fully Alink compatible and supports many additional options not found in Alink. Also is much faster than Alink and generates smaller executable files. Version 5.7 Author: Combined efforts of the Software Distillery
Browser	Updated version of the browser program released on disk number 18. This one has been ''manxified'', has scroll bars, and several bugs fixed. Author: Mike Meyer, enhancements and fixes by Bob Leivian
Btree	Routines to implement a B–tree algorithm and several accompanying tests, apparently derived from the single file version also included on this disk under ''Btree2''. Still buggy, but looks useful with a little work. Author: Richard Hellier, posted to usenet by Steve Jefferson
Btree2	Previous (original) version of btree routines. Author: Richard Hellier
Calendar	Appointment calendar that lets you visually add and update appointments. Can also run in background and remind you 15 minutes before any scheduled event. Author: Unknown, Amiga port by Bob Leivian
Less	Like Unix ''more'', only better, with forward and backward scrolling, searching and positioning by percent of file and line number, etc. Very useful! Author: Mark Nudelman, Amiga port by Bob Leivian
NewFonts	Set of 28 new Amiga fonts. Author: Bill Fischer
Pr	A background print utility with several options for letter quality compressed mode and line numbers. Also takes multiple files and wildcards. Author: Bob Leivian
Requester	A file name requester that looks like the one used in Deluxe Paint, and a sample program for using the requester. Author: Kevin Clague

README.listABI

This is disk 35 of the freely distributable AMIGA software library.

ASendPacket	Example program for sending multiple packets asynchronously to a dos handler, for those interested in implementing programs with asynchronous AmigaDos file I/O. Author: Phillip Lindsay
ConsoleWindow	Example program for finding the intuition pointer to an AmigaDos CON: or RAW: window, so you can do WindowToFront, graphics, and other interesting things. (Requires AmigaDos 1.2) Author: Andy Finkel and Robert Burns
DirUtil	Nice little program to wander around directory tree using a windowing interface and performing various operations on files. Author: Chris Nicotra, enhancements by Dave Jobusch
DirUtil2	Another variant of dirutil. Author: Chris Nicotra, enhancements by Ed Alford
FileRequester	A very nice file requester module for lattice C programs, along with a demo driver program. Author: Charlie Heath
MacView	Allows viewing of standard MacPaint picture files in either Amiga low resolution (320 x 200) or high resolution (640 x 400). Executable, sample pictures, and icons (version on disk 32 did not include pictures). Author: Scott Evernden
Plop	Short, simple, no–frills IFF reader program, and a sample picture made using a ray tracing algorithm. Author: Jim Kent
PopCLI	Provides a simple way of starting another CLI at any time without having to load workbench or exit whatever program you may be using. Also has a builtin screen saver mode that automatically blanks the Amiga console screen when there has been no input for a specified period of time. Author: John Toebes
QuickCopy	Three versions of quickcopy, a nice little full disk copier. Two of the versions are capable of making backup copies of ''protected'' Electronic Art's disks. Author: Dave Devenport
ScrollPf	Creates and displays a 400 by 300 by 2 bit plane playfield on top of a 320 by 200, 2 plane deep playfield, as a demo of dual playfield display. Author: Carolyn Scheppner
SendPacket	General purpose subroutine to send AmigaDos packets. Author: Phillip Lindsay
SpriteMaker	Program which lets you paint sprites and then converts the image into a C data structure to be used in programs. Allows interactive testing of the sprite as a pointer. Suggested shareware donation of $15 for source code and updates. Author: Ray Larson
Tracker	Program which converts a boot–load disk (I.E. a kickstart disk) into a group of files for electronic transmission and reassembly. Completely preserves the original disk structure so the target disk will be an exact duplicate of the original. Author: Brad (Lord Bradford) Wilson
TriClops	Very nice graphics oriented 3–D space invasion game. This was previously a commercial product which is being released into the public domain for promotional purposes (they are working on a multiuser, multimachine version). Author: Unknown, published by Geodesic Publications
Tsize	A simple utility to print directory tree sizes. Displays the total size of all files and subdirectories within a given directory. Author: Edmund Burnette
UnIfdef	Useful program for removing ifdef'ed sections from a file while otherwise leaving the file alone. Allows one source to be used as a porting base for many machines, without shipping the entire source to every source customer. (Preprocess for their machine and send them only the source for their specific machine). Author: Dave Yost
Vttest	Program to test compatibility of vt100–compatible terminals and terminal emulators. Requires the resources of a Unix system to test an Amiga hosted vt100 emulator. (I haven't yet found one that even comes close to passing this test!). Author: Per Lindberg

When you have a job to do,
get it done right. . .
J. Mostacci

The Transactor

The Transactor
The Tech/News Journal For Commodore Computers

Please send me **6** consecutive **Transactors** starting with the next issue!

☐ I'm subscribing to the magazine AND the disk, so please send me my **FREE T-Shirt** in the size indicated below.

☐ Canada **$15.00** ☐ U.S.A. **$15.00** U.S. ☐ Foreign **$21.00** U.S. ☐ Air Mail (Overseas only) **$40.00** U.S.

☐ Please send microfiche instead of magazines

☐ New Subscription ☐ Renewal (please include your Subscription Number from mailing label) _____

Name & Address New address? ☐

(please include your postal/zip code):

_____

_____

_____

_____

SAVE 29% OFF Cover Price

*** Ont. residents please add 7% PST**
† Prices are in U.S. Dollars

	$Cdn.	$U.S.		
☐	13.95*	13.95†	Transactor T's ☐S, ☐M, ☐L, ☐XL	
☐	17.95*	17.95†	Jumbo T-Shirt	
☐	14.95	14.95†	Transactor Book of Bits and Pieces #1	
☐	39.95*	39.95†	Transactor Microfiche Set (Vol4 + Vol5 + Vol6)	
☐	329.95*	199.95†	Volksmodem 12 with cable & CIS IntroPak (for ☐C64 User Port or ☐RS232 DB-25 Connector)	
☐	1035.00*	729.00†	AX1000 1 MEG Amiga board + S&H (see below)	
☐	1276.00*	899.00†	AX2000 2 MEG Amiga board + S&H (see below)	
☐	25.00	100.00†	* S&H encl. for each AX board, or ☐ Ship collect	
☐	79.95*	79.95†	The Toolbox (PAL and POWER for C64)	
☐	14.95	14.95†	The Complete Commodore Inner Space Anthology	
☐	59.95*	59.95†	1541 ROM Upgrade Kit	
☐	199.00*	129.00†	BH100 I/O Card with documentation	
☐	69.00*	45.00†	BH100–AD8 Module	
☐	239.00*	159.00†	BH100 Beginners Course	
☐	39.00*	25.00†	BH100–S Security System	

	$Cdn.	$U.S.		
☐	9.95*	9.95†	The TransBASIC Disk	
☐	129.95*	89.95†	The Micro Sleuth	
☐	39.95*	29.95†	Super Kit 1541	
☐	69.95*	59.95†	Gnome Speed Compiler	
☐	49.95*	39.95†	Gnome Kit Utility	
☐	29.95*	29.95†	Moving Pictures	
☐	59.95*	49.95†	Superpak 1.0	C64
☐	39.95*	29.95†	Pocket Writer	C64
☐	39.95*	29.95†	Pocket Planner	C64
☐	39.95*	29.95†	Pocket Filer	C64
☐	69.95*	59.95†	Superpak 1.0	C128
☐	59.95*	39.95†	Pocket Writer	C128
☐	59.95*	39.95†	Pocket Planner	C128
☐	59.95*	39.95†	Pocket Filer	C128
☐	19.95*	14.95†	Pocket Dictionary (included w/Superpaks)	

Note: See News BRK for details on above items
Please allow 6-8 weeks for delivery in U.S., 4-6 weeks in Canada

The Transactor Disk (1541/4040/MSD format)

☐ Please send 6 consecutive disks to correspond with my magazine subscription: **$45.00.***

Send these disks at **$8.95*** each.

☐ Disk 1 ☐ Disk 2 ☐ Disk 3 ☐ Disk 4 ☐ Disk 5
☐ Disk 6 ☐ Disk 7 ☐ Disk 8 ☐ Disk 9 ☐ Disk 10
☐ Disk 11 ☐ Disk 12 ☐ Disk 13 ☐ Disk 14 ☐ Disk 15 ☐ Disk 16

Transactor Back Issues: $4.50* each, MF $4.50* each (MF = Microfiche)

☐ Vol.4, Iss.01 ☐ MF
☐ Vol.4, Iss.02 ☐ MF
☐ Vol.4, Iss.03 ☐ MF
☐ Vol.4, Iss.04 - MF only
☐ Vol.4, Iss.05 - MF only
☐ Vol.4, Iss.06 - MF only
☐ Vol.5, Iss.01 ☐ MF
☐ Vol.5, Iss.02 ☐ MF
☐ Vol.5, Iss.03 - MF only
☐ Vol.5, Iss.04 - MF only
☐ Vol.5, Iss.05 ☐ MF
☐ Vol.5, Iss.06 ☐ MF
☐ Vol.6, Iss.01 ☐ MF
☐ Vol.6, Iss.02 ☐ MF
☐ Vol.6, Iss.03 ☐ MF
☐ Vol.6, Iss.04 ☐ MF
☐ Vol.6, Iss.05 ☐ MF
☐ Vol.6, Iss.06 ☐ MF
☐ Vol.7, Iss.01 ☐ MF
☐ Vol.7, Iss.02 ☐ MF
☐ Vol.7, Iss.03 ☐ MF
☐ Vol.7, Iss.04 ☐ MF
☐ Vol.7, Iss.05 ☐ MF

NOTE: Prepayment required. Purchase orders will be accepted ONLY if accompanied by payment.

☐ Cheque/MO. enclosed Cheque#_____ Dated____/____/____ Amount_____

☐ Visa ☐ MasterCard Acct. # _____ Expires ____/____

I use the following Commodore equipment:

☐VIC 20 ☐C 64 ☐C128 ☐Amiga ☐B128 ☐8032/96 ☐16 / +4
Disk Unit: ☐SFD1001 ☐1540/41 ☐4040 ☐8050 ☐8250 ☐1571

I use my equipment in the following environment:

☐Hobby ☐Business ☐Technical ☐Public School ☐High School ☐College/Univ. ☐CBM Dealer

☐ Please send dealer information for The Transactor.

12/86

fold along scores with correct reply card on outside – use tape, no staples please

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 390 BUFFALO, NY

POSTAGE WILL BE PAID BY ADDRESSEE

The Transactor
277 Linwood Avenue
Buffalo, NY, 14209-9990

Business Reply Mail
No Postage Stamp Necessary
if mailed in Canada

Postage will be paid by:

Canada Post Postes Canada
207

The Transactor
500 Steeles Avenue
Milton, Ontario, Canada
L9T 9Z9

Cover the Spectrum

with

PRISM SOFTWARE

Sixth Sense 64 — $39.95

It answers your phone, makes your calls, acts on both.

Sounds outrageous! It is! The Sixth Sense 64 modem software understands a macro language that operates based on the time of day, data received, internal counters or provided templates. Over 160 functions at your control!

- 700 virtual line screen • 16 macro keys
- 16 condition strings spot prompt/initiate responses
- Clock functions key operations/stamp incoming data

Sixth Sense 128 — $49.95

The spectrum of Prism expands to enhance your Commodore 128. With Sixth Sense 128 comprehensive modem control isn't a mission impossible.

Sixth Sense 128 is the most comprehensive modem control available. It operates based on the time of day, data received, internal counters or provided templates. Harness the explosive capabilities of Sixth Sense to do your next mission impossible.

- 800 line buffer/7,200 lines maximum with expanded RAM
- 20 active macros • Runs in 80 columns only
- 42 prewired command keys - 10 to wire your way!
- Line/screen editors • SEARCH/GOTO commands in buffer
- CompuServe "B" & XMODEM CRC/Checksum file transfer protocol

Dataquick 64 — $19.95

Extra! Extra! Calling all potential BBS and Exchange Operators!

Once again Prism Software offers the latest in software for the Commodore 64 user! Now with Dataquick 64 **you** can operate a BBS with 8 message bases and 10-25 messages per base. Included with Dataquick is the Lightning Exchange which makes multi-file transfers quick and easy.

Dataquick's EXTRAordinary features:

- Supports 1650/1660/1670, Westridge, Master Modem, Volksmodem 12 & Hayes compatible modems.
- Supports 1-4 disk drives. • Supports new Punter protocol.
- Control access to drive 10 & 11. Restricts to high level users.
- Secure - users see only what you let them see. 10 access levels for sysop control. Records hackers and leeches.
- Poll function - Storyboard - E-mail - Macros!
- Complete sysop support-documentation, maintenance programs, samples, setup programs & membership to private support line.

Lightning Exchange's shocking features:

- **Multi-file transfer**
- Supports same modems as Dataquick 64
- Built-in terminal • Supports 1-4 disk drives

Superkit 1541 — $29.95

version 2.0 by Marty Franz & Joe Peter

SINGLE NORMAL COPIER - Copies a disk with no errors in 1 minute. Corrects all disk errors.

DUAL NORMAL COPIER - Copies a disk in 33 seconds with a graphic/music display while working.

SINGLE NIBBLER - Nibble copies a protected disk in 1 minute.

DUAL NIBBLER - Nibbles a disk in 30 seconds and has a graphic/music display while working. It's capable of copying elongated headers, extra sectors and non-standard GCR.

FILE COPIER - Full screen display including buffer, starting track & sector, file being copied and revives deleted/corrupted files.

TRACK & SECTOR EDITOR - Capable of reading to track 40 and examines data under errors. Full editing capabilities in HEX, ASCII or text. An ML monitor is built-in.

GCR EDITOR - Allows examination of a disk in its raw format including the header, density, sync marks and non-standard GCR bytes. You can even examine a full track at a time. It's a great way to learn disk protection methods!

SUPER NIBBLER - The most powerful nibble available. It even detects and duplicates density changes automatically.

DISK SURGEON - This is what a parameter copier should be! It copies and places parameters on the disk. Now, over 400 parameters are included.

SUPER SCAN - Gives a video or printer display of errors and density on a disk in under 35 seconds.

SUPER DOS FAST LOADER - Loads 150 blocks in 10 seconds. It also includes an Auto-Boot maker.

All programs work with 1541/1571 single side drives made. All of the copiers are the fastest on the market and include directory options. The File Copier, Track & Sector Editor, Super Nibbler and Disk Surgeon use 1 or 2 drives and include device number change. All programs re-boot to main menu. SUPERKIT has an easy to use menu-driven operation! Version updates are $10. Parameter updates are $6.

Plus $3.00 Shipping/Handling Charge - $5.00 C.O.D. Charge

All of these programs come on a double-sided disk.

PRISM SOFTWARE

401 Lake Air Drive, Suite D Waco, Texas 76710

Orders/Tech Help (817) 751-0200

Dealers and distributors are welcome.

MASTERCARD & VISA ACCEPTED

SUPERKIT 1541 is for archival use only! We do not condone nor encourage piracy of any kind.

THE TIME SAVER

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only $8.95 Per Issue
6 Disk Subscription (one year)
Just $45.00
(see order form at center fold)

Also check out the TransBASIC Disk
Complete with 24 page manual, just $9.95!